



**Universidad
Andrés Bello®**

UNIVERSIDAD ANDRÉS BELLO

Facultad de Ingeniería

Escuela de Industrias

Prototipo de interfaz humano-maquina basado en AIML con capacidad de realizar tareas preprogramadas

Tesis para optar al grado “Licenciado en Ciencias de la Ingeniería”
y título “Ingeniero en Automatización y Robótica”.

Autor:

Manuel Ignacio Vallejos Sandoval

Profesor Guía: Néstor Palominos González

Santiago de Chile, 2018.



**Universidad
Andrés Bello®**

UNIVERSIDAD ANDRÉS BELLO
FACULTAD DE INGENIERÍA
ESCUELA DE INDUSTRIAS

INGENIERÍA EN AUTOMATIZACIÓN Y ROBÓTICA

DECLARACIÓN DE ORIGINALIDAD Y PROPIEDAD

Yo, **Manuel Ignacio Vallejos Sandoval**, declaro que este documento no incorpora material de otros autores sin identificar debidamente la fuente.

Santiago, Julio de 2018.

Firma del alumno

AGRADECIMIENTOS

A toda la gente que me colaboró con este proyecto, cada uno de los que se tomó un poco de tiempo ya sea para escucharme o ayudarme con las pruebas.

A mis compañeros de Universidad por ayudarme a resolver las dudas que tuve.

A mi profesor Néstor por apoyarme y guiarme durante este proceso.

DEDICATORIA

A mi familia por apoyarme y estar ahí siempre.

Gracias a toda la gente que siempre confió y creyó en mi para lograr mis metas y objetivos.

Dedicado a toda la gente que siempre me ha apoyado desinteresadamente

ÍNDICE GENERAL

AGRADECIMIENTOS	2
DEDICATORIA	2
ÍNDICE GENERAL	3
ÍNDICE DE FIGURAS	6
ÍNDICE DE TABLAS	7
RESUMEN	8
ABSTRACT	9
1 ANTECEDENTES GENERALES	10
1.1 INTRODUCCIÓN	10
1.2 OBJETIVOS	13
1.2.1 OBJETIVO GENERAL	13
1.2.2 OBJETIVOS ESPECÍFICOS	13
1.3 ORGANIZACIÓN Y PRESENTACIÓN DEL TRABAJO	14
2 MARCO TEÓRICO	15
2.1 ESTADO DEL ARTE	15
2.2 PLANTEAMIENTO DE LA PROBLEMÁTICA	17
2.3 TEXT TO SPEECH (TTS)	18
2.4 PROCESAMIENTO DE LENGUAJE NATURAL (NLP)	19
2.4.1 PART-OF-SPEECH TAGGING (POST)	21
2.4.2 PARSE TREES (ÁRBOLES DE ANALISIS SINTÁCTICO)	22
2.4.3 SEMANTICA	23
2.5 AIML EN ESPAÑOL	23
2.6 TECNOLOGÍA AUTOMATIC SPEECH RECOGNITION (ASR)	24
2.7 TEORÍA DE MUESTREO DE NYQUIST	25
2.8 ASISTENTES VIRTUALES ACTUALES	26
2.9 HERRAMIENTAS UTILIZADAS	27
2.9.1 AIML (Artificial Intelligence Markup Language)	27
2.9.2 GOOGLE CLOUD SPEECH RECOGNITION	39
2.9.3 RASPBERRY PI 3 MODEL B	40
2.9.4 RASPBIAN S.O.	41
2.9.5 FESTIVAL TTS	42

2.9.6 ZX-THERMOMETER.....	43
2.9.7 I2C ADS1115.....	44
2.10 CARTA GANTT	45
3 DESARROLLO DEL PROTOTIPO	46
3.1 DIAGRAMA DE BLOQUES	46
3.2 DIAGRAMA DE FLUJO	48
3.3 PROCESAMIENTO DE LA ENTRADA Y SALIDA DE AUDIO	49
3.4 PROCESAMIENTO EN AIML.....	50
3.5 CONFIGURACIÓN DEL BOT.....	51
3.5.1 SECCION DEL BOT	52
3.5.2 CONFIGURACIÓN DE ORTOGRAFÍA.....	54
3.5.3 CONFIGURACIÓN DEL CEREBRO DEL BOT	55
3.6 CÓDIGO EN PYTHON 3	68
3.6.1 RECONOCIMIENTO DE VOZ	69
3.6.2 CONVERSIÓN DE TEXTO A VOZ	70
3.6.3 CONFIGURACIÓN DE GPIO	71
3.6.3 CONSOLA DEL CHATBOT	73
3.7 SET DE AIML	75
3.7.1 AIML PRINCIPAL	76
3.7.2 AIML PARA AGREGAR CATEGORÍAS	77
3.7.2 BADANSWER AIML	78
3.7.3 AIML PARA USAR GPIO.....	80
4 ANALISIS DE RESULTADOS.....	82
4.1 PARÁMETROS DE PRUEBA.....	82
4.2 PRUEBAS DE INICIALIZACIÓN	83
4.3 ENTRENAMIENTO DEL CHATBOT	85
4.4 PRUEBAS CUANTITATIVAS	86
4.4.1 GRÁFICOS DE LOS RESULTADOS.....	88
4.4.1 PRUEBAS CUALITATIVAS.....	94
4.4.2 RESULTADOS EVALUACIÓN	95
5 DISCUSIÓN	96
5.1 RESULTADOS OBTENIDOS	96

5.2 RECONOCIMIENTO DE VOZ.....	97
5.3 INTERACCIÓN DEL CHATBOT.....	98
5.4 MEDICIÓN DEL CHATBOT	99
5.5 ALCANCES Y LIMITACIONES	100
6 CONCLUSIONES	101
7 GLOSARIO	103
8 REFERENCIAS	104
9 ANEXOS.....	106

ÍNDICE DE FIGURAS

Figura 1 Diagrama de Síntesis de voz	18
Figura 2 Parse Tree en inglés.....	22
Figura 3 Parse Tree en español.....	22
Figura 4 Diagrama de Speech to Text	24
Figura 5 Proceso de muestreo.....	25
Figura 6 Resultados de comparación entre asistentes virtuales actuales.....	26
Figura 7 Logotipo de A.L.I.C.E.....	28
Figura 8 Graphmaster de 5 categorías	38
Figura 9 Funcionamiento de Google Cloud Speech API.....	39
Figura 10 Detalles Raspberry PI 3 Model B	40
Figura 11 Pinout Rasperry Pi 3 Model B.....	40
Figura 12 ZX-Thermometer sensor de Temperatura	43
Figura 13 Pinout de ZX-Thermometer	43
Figura 14 Conversor Analogo Digital de 16 bits de resolución ads1115.....	44
Figura 15 Carta Gantt	45
Figura 16 Diagrama de Bloques	46
Figura 17 Diagrama de Flujo	48
Figura 18 Pattern Path incluyendo <that> y <topic>	50
Figura 19 Ejecutable .bash del bot.....	51
Figura 20 Configuración de archivos del bot.....	60
Figura 21 Organización de carpetas del bot	60
Figura 22 Diagrama del código de Reconocimiento de Voz	69
Figura 23 Código de FestivalTTS	70
Figura 24 Código para utilizar GPIO como output	71
Figura 25 Diagrama del código de GPIO	71
Figura 26 Conexión de ventilador 5V.....	72
Figura 27 Configuración del ADC en Python 3	72
Figura 28 Conexión de I2C ads1115 y ZX-Thermometer	72
Figura 29 Diagrama del código de la consola del Chatbot.....	74
Figura 30 Archivos AIML del Chatbot	75
Figura 31 Graphmaster de las categorías usadas por el Chatbot.....	75
Figura 32 Pattern Path de badanswer.aiml.....	79
Figura 33 Gráfico del tiempo de inicialización.....	83
Figura 34 Gráfico de la Pregunta 1	88
Figura 35 Gráfico de la Pregunta 2	89
Figura 36 Gráfico de la Pregunta 3	89
Figura 37 Gráfico de la Pregunta 4	90
Figura 38 Gráfico de la Pregunta 5	90
Figura 39 Gráfico de la Pregunta 6	91
Figura 40 Gráfico de la Pregunta 7	91
Figura 41 Gráfico de la Pregunta 8	92
Figura 42 Gráfico de las respuestas agregadas	93

ÍNDICE DE TABLAS

Tabla 1 Tags básicos de AIML 1.0.....	31
Tabla 2 Acción recursiva de SRAI	32
Tabla 3 Tags complejos de AIML 1.0.....	33
Tabla 4 Tags condicionales de AIML 1.0	34
Tabla 5 Tags incorporados en AIML 2.0	36
Tabla 6 Configuración de bot 1	52
Tabla 7 Configuración de bot 2	53
Tabla 8 Configuración de ortografía.....	54
Tabla 9 Configuraciones del cerebro	55
Tabla 10 Configuración de overrides	56
Tabla 11 Configuración de defaults	57
Tabla 12 Configuración de nodos	58
Tabla 13 Configuración de binarios	59
Tabla 14 Configuración de archivos aiml	61
Tabla 15 Configuración alternativa de errores	62
Tabla 16 Configuración de sets	63
Tabla 17 Configuración de maps	64
Tabla 18 Configuración de archivos rdf	65
Tabla 19 Configuración de archivos substitutions.....	66
Tabla 20 Configuración de archivos processor	67
Tabla 21 Librerías necesarias para Python3	68
Tabla 22 Medidas de tiempo de inicialización.....	83
Tabla 23 Pruebas de tiempo de inicialización	84
Tabla 24 Preguntas de prueba del chatbot	86
Tabla 25 Calificación cualitativa del bot	95

RESUMEN

“La inteligencia artificial está evolucionando con nuevas y mejores innovaciones todos los días. Pero fundamentalmente la I.A. siempre ha intentado perseguir y obtener rasgos de inteligencia humana.” (Bishwajeet, 2015). En el mundo actual es inevitable la evolución tecnológica hacia lo simple y automático y la expansión de esto a los distintos ámbitos de los negocios. Hoy en día existen aplicaciones que son capaces de dar respuestas satisfactorias a los usuarios a través de chat, pero aun así las capacidades de respuesta siguen siendo limitadas.

Este proyecto presenta una investigación y desarrollo de un prototipo capaz de impulsar las capacidades de las actuales interfaces humano-maquina, las cuales son bastante escasas y limitadas en su funcionamiento.

Haciendo uso de herramientas como el lenguaje AIML, Python 3 y Google Speech API se logró una capacidad de funcionamiento más amplia y efectiva para estas interfaces de respuesta automática. Para esto se implementó una interfaz de comunicación capaz de recibir solicitudes realizadas por voz del usuario y dar una respuesta acorde, ya sea por medio del chat, por voz, o por alguna aplicación interna de este mismo.

También se realizaron pruebas para evaluar el grado de error que este tiene en sus con una metodología objetiva y subjetiva. “La evaluación de naturalidad de la conversación se ha dividido en dos partes: una parte objetiva, que toma en cuenta la cantidad de respuestas erróneas y correctas y otra subjetiva, que se enfoca en la experiencia del usuario.” (Quintero, 2015). La evaluación de esta interfaz resultó ser positiva obteniendo un 80% de precisión en el reconocimiento de voz y un 90% de precisión en el procesamiento de AIML. De esto se determinó estas interfaces son benéficas siempre y cuando se tomen todas las medidas y se ajusten los parámetros de manera correcta. De no ser así las capacidades de procesamiento se pueden ver mermadas considerablemente.

Palabras clave: AIML, Chatterbot, Google Speech API, Reconocimiento por voz, Interfaz humano-maquina, automatico.

ABSTRACT

“Artificial Intelligence is coming up with new and greater innovations every day. But fundamentally, A.I. has always tried to pursue and attain intelligent human traits” (Bishwajeet, 2015). It is a fact that nowadays there are totally functional applications which are capable of giving satisfying responses to the users, frequently by chat communication, even so, the capabilities of answer are still limited.

This Project presents an investigation and development of a prototype capable of boosting the capabilities of the current human-machine interfaces, which are very poor and limited in its functionality. Using tools like Chatterbot, AIML language and Google Speech API the performance boosting was accomplished in an effective and wider form. For this purpose an interface was developed with the capacity to receive voice request of the user and give an answer by chat, voice or an internal application.

The margin of error of the responses to the user can also be evaluated, one of the ways is using an objective and a subjective methodology. “The evaluation of the conversation’s naturalness has been Split into two parts: an objective part, that measures the correct and incorrect answers, and a subjective part, that focuses on the user's experience”. (Quintero, 2015). The evaluation of this interface ended being positive, obtaining an 80% of accuracy in speech recognition and a 90% of accuracy in AIML processing. After this results it was determined that this interfaces can be beneficial but only if all the measurements are taken and the parameters are adjusted correctly. If not, the processing capabilities can be decreased considerably.

Keywords: AIML, Chatterbot, Google Speech API, voice-made request, Human-Machine Interface, automatic.

1 ANTECEDENTES GENERALES

1.1 INTRODUCCIÓN

A lo largo de los años y luego de que el humano dejó relativamente de lado la evolución biológica propiamente tal, la evolución se ha enfocado en el área tecnológica, creando herramientas, complementos, accesorios y diferentes tipos de artefactos para poder satisfacer las necesidades humanas y también generar comodidades. “Rápidos avances tecnológicos fueron requeridos para lograr satisfacer el incremento de demanda por un estilo de vida fácil y confortable. Los sistemas de automatización domóticos están ideados para proveer de conveniencia, energía, eficiencia, seguridad y confort.” (D Celebre, 2015).

Como ya desde hace tiempo se viene apuntando en el desarrollo de la tecnología, cada vez las tareas y/o procesos se realizan de manera más autónoma, reemplazando así mecanismos obsoletos e incluso mano de obra. Aun así estas tareas constan de acciones netamente mecánicas dentro de industrias o en terreno donde solo están en contacto directo con el proceso a controlar y donde la tarea a realizar nunca involucrará tener contacto con una persona de manera directa para solucionar su problema.

Hoy en día los avances en el contacto de la máquina con el usuario están muy avanzados, ya sea por medio de controles o Interfaces Humano-Máquina (HMI), pero en la mayoría de los casos está limitado a aplicaciones de carácter industrial. “Hoy en día las tecnologías de conversación están disponibles comercialmente para un rango de tareas limitado pero bastante interesante. Aun así, el desarrollo de estas tecnologías hace a las máquinas capaces de responder al diálogo de manera confiable y de esta manera es posible explotar los beneficios de la tecnología de la información para llegar a los usuarios corrientes.” (Ganesh, 2015). Esta capacidad de conversación se aprovecha en este proyecto mediante el uso de un Chatbot que puede cumplir necesidades de los usuarios a los cuales no están en la misma situación que la mayoría.

Los Chatbots o Chatterbots no son nada nuevo en la actualidad, puesto que existen hace mucho tiempo y se ocupan generalmente en labores pequeñas. A pesar de tener usos simples, el potencial de estos entes generados por computadora se ha ido ampliando cada vez más con el paso de los años, pudiendo abarcar bases de conocimiento y análisis de patrones cada vez más precisos. Utilizando las capacidades de un Chatbot y otros programas se puede crear una interfaz capaz de interactuar efectivamente con el usuario y además a través de un canal que abre las posibilidades a cualquier persona.

En este proyecto se presenta una solución capaz de incluir a personas discapacitadas, postradas o con algún problema de movilidad ya que se hace uso de un Chatbot capaz de comunicarse por voz como interfaz entre el usuario y la máquina para lograr cumplir las funciones que los usuarios no pueden debido a su condición.

En la actualidad las aplicaciones de conversación están bastante desarrolladas por algunas empresas de gran calibre como Cortana de Microsoft, Siri de Apple y Google Assistant. Estos programas incorporados inicialmente a los dispositivos móviles (tablets y smartphones) estaban diseñados para cumplir o acatar necesidades básicas por medio de la voz y memorizando los patrones frecuentes del usuario. Los programas diseñados por estas empresas crecieron rápidamente y cada vez aumentan sus capacidades, pero lamentablemente este tipo de aplicaciones se expanden hacia productos de su propia marca o corporación, haciendo que se adapten a este tipo de productos más que a las necesidades precisas del usuario. Debido a esto usar un asistente virtual de las compañías grandes supondría un costo mucho mayor en la producción de un Chatbot.

La aplicación desarrollada en este proyecto consta de un agente conversacional o Chatbot basado en AIML 2.0 (Artificial Intelligence Markup Language) adaptado al lenguaje español y con una base de conocimientos ampliable para dar respuestas. Además el usuario puede hablarle a este ente para solicitarle tareas, ya que usando Google Cloud Speech API y FestivalTTS se puede lograr una comunicación por voz. Este Chatbot esta montando en una Raspberry Pi 3, la cual da la funcionalidad de poder interactuar y generar acciones a través de sus pines GPIO y relés, de esta forma el usuario puede prender o activar dispositivos solo mediante el uso de la voz.

1.2 OBJETIVOS

1.2.1 OBJETIVO GENERAL

Se desarrollará un prototipo de interfaz humano maquina basado en AIML dentro de una plataforma Raspberry PI que sea capaz de comunicarse por voz y generar acciones determinadas para el usuario usando un reconocedor de voz y un sintetizador de voz.

1.2.2 OBJETIVOS ESPECÍFICOS

- Desarrollar una base de conocimientos de AIML 2.0 en español que sea eficiente y capaz de agregar nueva información de manera tal que cumpla con los requisitos del usuario.
- Implementar esta aplicación en una plataforma de Raspberry PI capaz de interactuar con el entorno a través de sus pines para generar acciones determinadas.
- Implementar una interfaz de comunicación por voz usando un reconocedor de voz y un sintetizador de voz.
- Realizar un análisis del tiempo de respuesta y margen de error en las respuestas de la aplicación y medirlo mediante evaluaciones cuantitativas y cualitativas.

1.3 ORGANIZACIÓN Y PRESENTACIÓN DEL TRABAJO

La estructura del desarrollo de este proyecto se encuentra distribuida de la siguiente manera a través del documento:

En el Capítulo 1 se introduce al lector al tema mediante la introducción de la problemática, la solución a esta y los objetivos que se pretenden cumplir en este proyecto.

En el Capítulo 2 se abordan los elementos a utilizar durante el proyecto, como AIML, Reconocimiento de voz y Síntesis de voz explicando el funcionamiento y el trasfondo de cada uno. También se explica el uso de estos para lograr resolver la problemática de las comunicaciones entre humano y máquina que existen hoy en día.

En el Capítulo 3 se explica el desarrollo del proyecto usando los elementos mencionados para modelar la interfaz del chatbot y explicando la conexión entre cada uno para llegar a la solución final.

En el Capítulo 4 se analizarán los resultados obtenidos por el prototipo, evaluando la precisión y eficacia de este para obtener un resultado cuantitativo de nuestro proyecto.

En el Capítulo 5 se realiza la discusión de los resultados obtenidos para ver la calidad de las pruebas y discutir sobre el desempeño del Chatbot.

Por último en el Capítulo 6 se realizan las conclusiones correspondientes al desarrollo del proyecto y su resultado final, así como también evaluar las distintas posibilidades que podría tener a futuro si se quiere aumentar la potencia del proyecto añadiendo más elementos.

2 MARCO TEÓRICO

2.1 ESTADO DEL ARTE

Los servicios hacia el cliente se han ido optimizando más y más a medida que avanza el tiempo y junto con este la tecnología usada para estos, partiendo desde la interacción directa del personal de trabajo con el usuario hasta la atención telefónica, por chat, o por e-mail. A pesar de estos avances, aún sigue habiendo un punto que optimizar: la necesaria interacción del personal de trabajo, ya que lógicamente este debe comunicarse con el usuario para poder entregarle el servicio requerido. Para obtener una solución a esta problemática es que se está evaluando y practicando el uso de Chatbots o agentes conversacionales, los cuales tendrían la capacidad de atender al usuario y proporcionarle las respuestas y/o información que este necesite de manera rápida, concisa y además con la capacidad de interactuar con varios usuarios a la vez, transformando esta tecnología en un impulso gigante para la industria o empresa que lo utilice.

“Poner un centro de soporte a clientes funciona bien para la audiencia pequeña-mediana, pero para empresas masivas como sistemas de gobiernos electrónicos los sistemas de soporte difícilmente pueden manejar la demanda de los usuarios.” (Mahapatra, 2012).

Gracias a la rápida evolución de la tecnología de esta índole ya existen varias implementaciones de este tipo. “Los avances en Reconocimiento Automático de Voz (ASR), Traducción Automática (MT) y la síntesis de Texto a Voz (TTS) han conllevado a lograr una traducción de Voz a Voz (S2S) con una calidad aceptable.” (Gopi, 2015). Los usos más actuales de este tipo de interfaces virtuales se pueden encontrar en plataformas online, los cuales son asistidos y apoyados por ALICE, una Fundación de Inteligencia Artificial que promueve el desarrollo de su Software y de AIML (Artificial Intelligence Markup Language) gratis. Entre estas aplicaciones están agentes conversacionales para ayudar a aprender y practicar un segundo lenguaje, así como también consejos propios de usuarios para mejorar el servicio y para resolución de dudas.

Una de las principales diferencias entre Chatbots radica en la capacidad de este para añadir conocimiento a su base de datos como si este aprendiese solo. Dentro de esta categoría se encuentran los más potentes en su campo, como Siri, Cortana y Google Assistant, pero también hay aplicaciones libres que permiten crear un Chatbot capaz de aprender, como es el caso del lenguaje AIML que a pesar de ser un programa abierto y de menor calibre en su versión 2.0 se le ha integrado la función de “aprender”.

La capacidad de aprender de esta nueva versión de AIML se basa en el uso del Natural Language Processing en conjunto con los tags de este mismo. Usando los tags se logra que el bot procese el lenguaje natural hablado por el usuario y agregue la información en forma de nuevas categorías, esto identificando los patrones que han sido determinados previamente para añadir el nuevo conocimiento. Estos datos pueden ser agregados por el usuario por medio de los patrones determinados o añadidos automáticamente dependiendo de la aplicación que se le de al Chatbot. Las nuevas categorías pueden ser agregadas y usadas durante la misma ejecución.

2.2 PLANTEAMIENTO DE LA PROBLEMÁTICA

El desarrollo de los chatbots está avanzando de manera progresiva a medida que las tecnologías se expanden, es por esta razón que siempre se pueden generar implementaciones que sean capaces de mejorar o complementar las cualidades pertenecientes a estos entes computacionales. Ya que los chatbots se comunican por escrito es necesario usar una interfaz con teclado y pantalla para estos, pero implementándole un canal de voz a la interfaz se facilita la comunicación entre el usuario y la maquina, permitiendo a incluso a usuarios discapacitados realizar tareas y acciones a través del Chatbot.

La implementación presentada en este proyecto consiste en el uso de AIML versión 2.0 instalado en un circuito integrado como la Raspberry PI 3 Model B con capacidad para comunicarse por medio de voz con el usuario. Este medio de comunicación será usado por el Chatbot que será implementado en una plataforma semi portátil para permitir a las personas con problemas de movilidad o con discapacidades generar acciones tales como revisar datos o encender dispositivos a través de esta interfaz.

La aplicación de un Chatbot en un ambiente no controlado y en contacto con distintas variables y personas es sin duda un gran cambio a la mayoría de los usos que habitualmente se usan. Esta situación le permitirá al Chatbot poder desarrollar nuevos conocimientos y aprender por medio de sus algoritmos ya que el estar en contacto con personas no expertas en el ámbito permite que las necesidades desde el punto de vista los de usuarios sean consideradas por el bot.

“(..) Los servicios robóticos son enviados a hogares donde no-expertos en robotica (ancianos y cuidadores) tienen ideas de que es lo que los robots deberían hacer, algoritmos de control pueden ser desarrollados para cuantificar estas manipulaciones” (Cao, 2013)

2.3 TEXT TO SPEECH (TTS)

Text to speech es una tecnología de ayuda para los usuarios que tienen dificultades para leer. Este tipo de software se encarga de reproducir mediante audio un párrafo de texto de cualquier documento.

Las voces provenientes de los TTS son generadas por computadora, por lo que puede ser acelerada o ralentizada para mejorar su comprensión. La calidad de la voz varía dependiendo del sintetizador que se use.

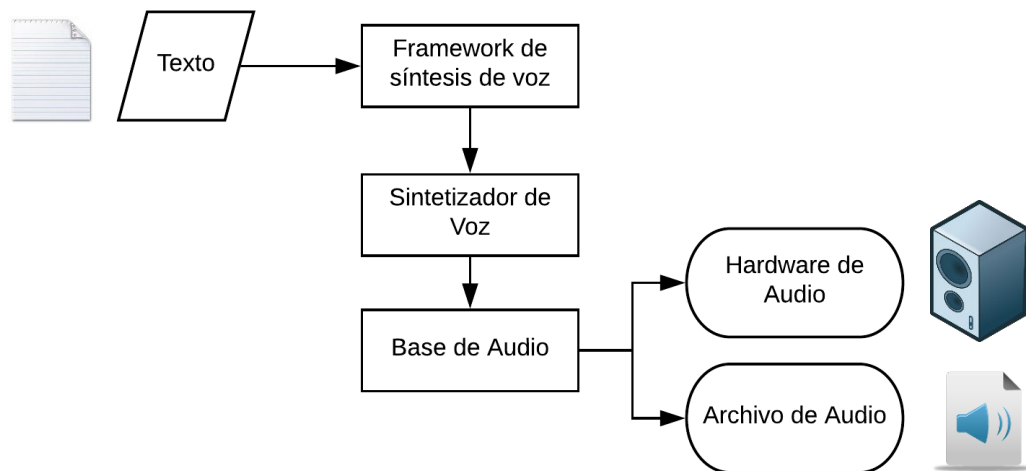


Figura 1 Diagrama de Síntesis de voz

Esta tecnología está ampliamente expandida hoy en día, por lo que cada vez se va estudiando más la calidad de la interpretación para lograr que sea lo más parecido a una voz humana.

“Típicamente después de la presentación de un estímulo acústico, el proceso de criterio de calidad del discurso humano involucra dos sub procesos, llamados perceptual y valoración”. (Gupta, 2017).

2.4 PROCESAMIENTO DE LENGUAJE NATURAL (NLP)

El procesamiento de lenguajes naturales (PLN) o *Natural Language Processing* (NLP) en inglés, es un campo de las ciencias de la computación, inteligencia artificial y lingüística que se basa en el estudio de las interacciones entre el humano y las computadoras. A diferencia de los lenguajes de programación convencionales utilizados comúnmente, el PLN trata de establecer un medio común tanto para el usuario como para la máquina, utilizando un lenguaje que ambos puedan entender y de esta manera lograr una organización eficiente de la información. Este tipo de lenguaje se utiliza principalmente para traducciones automáticas y aplicaciones de Chatbot de respuesta automática, así como también el Machine Learning y otras interfaces de comunicación humano máquina.

“El procesamiento del habla es un área de aplicación del NLP que involucra la codificación y decodificación de señales auditivas articuladas por los humanos para producir una letra, palabra, o sentencia.” (Shivakumar, 2016).

La función principal del NLP es lograr interpretar de manera correcta la oración y extraer la mayor cantidad posible de información para poder dar una traducción o una respuesta que sea acorde a la entrada ingresada por el usuario. La interpretación es la parte más compleja puesto que en el lenguaje cada palabra puede tener más de un mismo significado y las oraciones pueden tener doble sentido o ambigüedades, cosas que una máquina no podrá interpretar de la misma forma que un humano.

Para poder enseñarle el lenguaje a una máquina, los lingüistas computacionales recurren a las diferentes áreas de la lingüística:

-La **morfología** se ocupa de la composición de las palabras y de sus relaciones con otras palabras.

-La **sintaxis** define el modo en que las palabras se unen para formar frases.

-La **semántica** hace referencia tanto al significado de las palabras como al de los grupos de palabras.

-La **pragmática** se hace eco del contexto en el que se llevan a cabo las locuciones lingüísticas.

-La **fonología** se ocupa de la estructura fonética del lenguaje hablado y es importante para el reconocimiento de la voz.

Estas áreas de la lingüística son distintas para cada lenguaje, ya que en cada dialecto las palabras como verbos, sustantivos y adjetivos se ordenan de manera distinta.

Si hacemos una comparación de la forma estructural de las oraciones en inglés y español se puede ver claramente la diferencia.

The Little boy Runs Quickly

En el caso del inglés la estructura se compone de la siguiente manera

Artículo + Adjetivo + Sustantivo + Verbo + Adverbio

El niño pequeño corre rápidamente

A diferencia del español donde tenemos que la estructura es

Artículo + Sustantivo + Adjetivo + Verbo + Adverbio

Esta diferencia en la formación de la frase es de vital importancia para el procesamiento de las máquinas, ya que afecta directamente la forma de analizar la oración.

2.4.1 PART-OF-SPEECH TAGGING (POST)

Existen varios procesos de NLP que se enfocan en el análisis de una oración como el etiquetado gramatical (Part-Of-Speech Tagging). En este caso se trata de analizar la función de cada palabra por separada tratando de evitar cualquier ambigüedad que se pueda malinterpretar en ellas. Para esto se recurre a diversos métodos, los más antiguos se basan en textos extensos como el Brown Corpus o el British Nacional Corpus, estos textos están formados por millones de palabras etiquetadas y de las que se pueden deducir normas de aprendizaje para el etiquetado de palabras. Los programas más nuevos emplean algoritmos de autoaprendizaje, lo que significa que sobreentienden las normas de los corpus existentes de manera automática.

2.4.2 PARSE TREES (ÁRBOLES DE ANALISIS SINTÁCTICO)

En este procesos se emplean los conocimientos extraídos de la sintaxis para entender la estructura de las oraciones. En la lingüística computacional se usa la división en sintagmas los cuales pueden ser sintagmas nominales que están formados por un nombre propio o por un sustantivo o artículo, o los sintagmas verbales cuyo núcleo es un verbo.

En ingles se usa el término de **parsing** a la división de oraciones en sintagmas y en consecuencia los arboles de análisis sintácticos reciben el nombre de **parse tres**. Cada idioma tiene una gramática propia, lo que significa que los sintagmas se forman de manera distinta la jerarquía de sintagmas funciona de forma diferente. La gramática de un idioma se puede programar a mano en programas informáticos o se puede aprender mediante corpus de texto, que pueden servir para dejar constancia de la estructura de las oraciones.

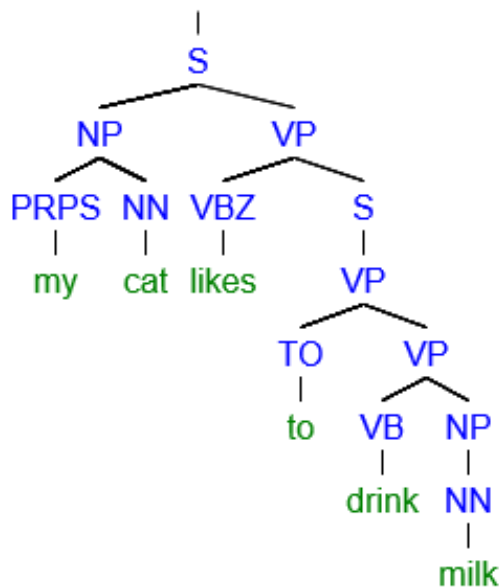


Figura 2 Parse Tree en inglés

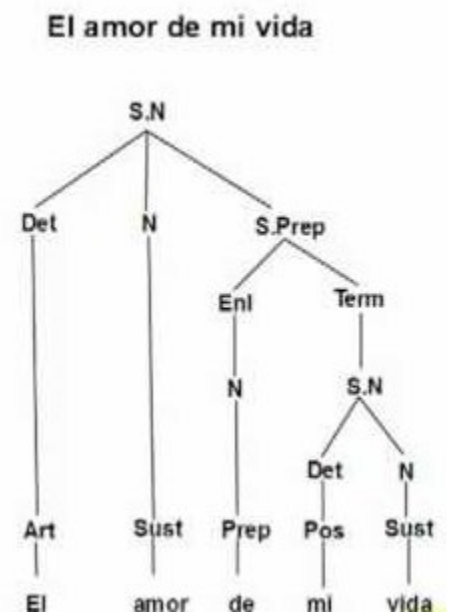


Figura 3 Parse Tree en español

Análisis sintáctico de la frase “My cat likes to drink milk” La oración (S) se divide en Sintagma Nominal(NP) y Sintagma Verbal(VP) y esta a su vez se subdivide en otros sintagmas.

2.4.3 SEMANTICA

Este proceso se encarga de determinar el significado de las palabras para que el ordenador pueda reconocer las ambigüedades de los términos usados. En la mayoría de los casos se hace un análisis del texto que lo precede o que le sigue para determinar si está refiriendo a un significado o al otro.

2.5 AIML EN ESPAÑOL

Dentro de los NPL AIML está bastante expandido y es ampliamente utilizado por usuarios del internet gracias a ser gratis y código libre. Esto también ha hecho que surjan varios proyectos propios, así como colaboraciones entre autores, de esta forma se han ido traduciendo los sets de AIML original del inglés al español y también han aparecido sets nuevos para implementar bots.

A pesar de contar con los sets en español el funcionamiento de AIML está determinado por otros factores, tales como el parse tree, los maps, los sets y diversos archivos de configuración, los cuales están principalmente ideados para el idioma inglés.

Dado que la versión de AIML 2.0 es relativamente nueva hay poco material en español para trabajar, por lo que es necesario hacer los ajustes y adaptaciones necesarias para lograr un procesamiento óptimo de este lenguaje.

2.6 TECNOLOGÍA AUTOMATIC SPEECH RECOGNITION (ASR)

“La tarea de la tecnología ASR es de reconocer a grandes rasgos el habla y proveer una salida de texto de lo que fue hablado.” (Tsontzos, 2011).

La tecnología ASR (Automatic Speech Recognition) o Reconocimiento de Voz Automático en español, es bastante conocida y utilizada ampliamente en la actualidad gracias a su manera sencilla de simplificar tareas y ser más amigable con el usuario.

Estos últimos años la tecnología ASR ha incrementado su uso en servicio al cliente de grandes empresas y también para organizaciones gubernamentales y agencias. “Recientemente, los sistemas TTS se han usado en muchos lugares. Por ejemplo, se pueden escuchar sonidos de TTS como anuncios en la estación, en el aeropuerto o en el bus.”(Suro, 2016). Esta tecnología permite reducir la interacción del usuario a respuestas de SI o NO y es capaz de entregar datos de dirección o números de teléfono, como efecto secundario produce que se reduzcan el número de llamadas en los call centers evitando la saturación de estos.

A pesar de ser una tecnología que lleva algún tiempo en el mercado aún falta perfeccionarla, ya que posee problemas para dar respuestas con usuarios con acento distinto o combinaciones de palabras en distintos idiomas y también el canal usado para comunicarse puede afectar la interpretación. El costoso valor de la implementación también es una barrera para las empresas de un nivel más pequeño que requieran implementarla.

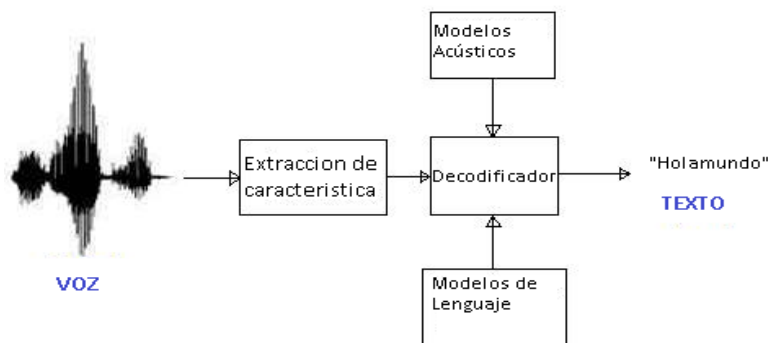


Figura 4 Diagrama de Speech to Text

2.7 TEORÍA DE MUESTREO DE NYQUIST

La tasa o frecuencia de muestreo corresponde al número de muestras que se toman de una señal continua por unidad de tiempo. El objetivo de este muestreo es poder traducir la señal continua sin pérdida de información ni errores, es decir, que el proceso sea reversible matemáticamente.

El Teorema de Nyquist propone que para que se pueda llevar a cabo una reconstrucción satisfactoria de la señal la frecuencia de muestreo de la señal continua debe ser al menos 2 veces el valor de la frecuencia de la señal original.

La frecuencia más alta es llamada Razón de Muestro de Nyquist y la mitad de este valor, correspondiente a la frecuencia de la señal original, se conoce generalmente como frecuencia de Nyquist.

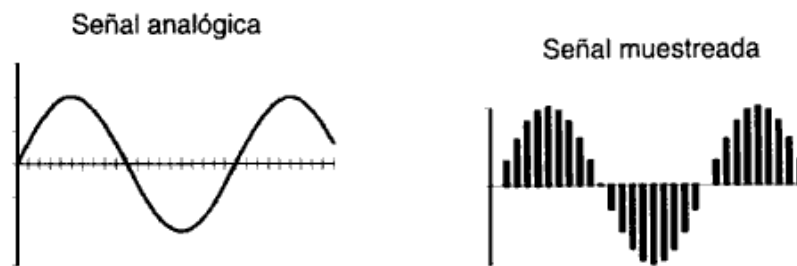


Figura 5 Proceso de muestreo

Las frecuencias audibles por el oído humano van desde los 20Hz a los 20000Hz y la voz se encuentra en un rango de frecuencia de 350Hz a 3000 Hz pero hay algunos sonidos entre los 4000Hz y 8000Hz.

Es importante destacar que la frecuencia de muestreo usada debe ser la correcta ya que si se utiliza una frecuencia distinta a la establecida por el teorema de Nyquist (menor a la mitad de la frecuencia de muestreo) se producirá un efecto Aliasing el cual impide la correcta reconstrucción de la señal.

2.8 ASISTENTES VIRTUALES ACTUALES

Tomando como referencia los asistentes virtuales mas desarrollados en la actualidad tenemos a Siri (Apple), Cortana (Microsoft) y Google Now (Google). Estos tres asistentes han estado compitiendo entre si durante los últimos años para determinar cuál es el más efectivo a la hora de desempeñar sus labores de ayuda.

Las mediciones utilizadas para comparar estos asistentes consisten en pedirles a usuarios al azar que ejecuten una lista de instrucciones en sus dispositivos utilizando los distintos asistentes.

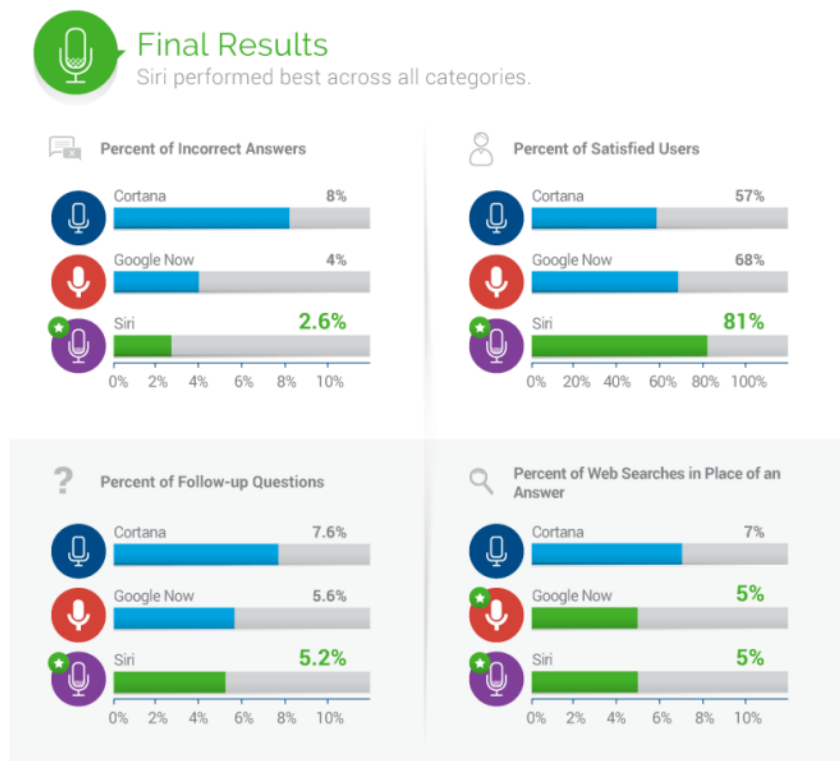


Figura 6 Resultados de comparación entre asistentes virtuales actuales

Actualmente Siri ha demostrado tener la mayor cantidad de respuestas correctas además de ser el más preciso a la hora de entregarlas.

Es importante destacar que en la medición de estos asistentes se evalúa la ejecución y manejo de aplicaciones en los dispositivos además de la comunicación por voz con el usuario.

2.9 HERRAMIENTAS UTILIZADAS

2.9.1 AIML (Artificial Intelligence Markup Language)

2.9.1.1 HISTORIA

XML (Extensible Markup Language), traducido como “Lenguaje de Marcado Extensible” es un meta-lenguaje que define reglas para codificar documentos tanto legibles para el humano como para las maquinas. Por esta razón es la base para el lenguaje AIML.

El dialecto XML de nombre AIML fue desarrollado por Richard Wallace y una compañía de software gratuito de nivel mundial entre 1995 y 2002. AIML formo la base para lo que era un ELIZA avanzado, y lo llamó “A.L.I.C.E.” (Artificial Linguistic Internet Computer Entity), el cual gano el “Loebner Prize Competition in Artificial Intelligence” tres veces. Este Premio Loebner se otorga al ganador de una competencia anual de chatterbots la cual consiste en un Test de Turing standard. “A principios de los ‘50, Alan Turing propuso el famoso Test de Turing, uno de los principales retos en el campo de la Inteligencia Artificial. Este test intenta definir lo que puede ser la inteligencia proporcionada por un ordenador, y al mismo tiempo, la posibilidad de que las máquinas puedan imitar el razonamiento humano.” (López De Luise, 2016). En cada ronda uno de los jueces mantiene una conversación por texto con un Chatbot y con un ser humano, mediante las respuestas el juez debe determinar cuál es cual. El Chatbot que sea el más humanizado será el ganador.

El test de Turing, también conocido como “The Imitation Game” estaba propuesto para determinar si un ente computarizado era capaz de convencer a otra persona de que este fuese un ser humano común. Turing estableció las condiciones de este test en sus papeles de 1950 entre las cuales la principal es que el contacto solo debe ser mediante texto. La primera fase consta de un Juez el cual estará haciendo preguntas hacia dos sujetos, uno femenino y uno masculino, sin saber tener conocimiento sobre cual es de cual sexo mientras que los sujetos tratan de inducir equívocamente al juez sobre su información. La segunda fase es la que involucra un ente computarizado, pues se reemplaza uno de los dos sujetos por un Chatbot el cual tendrá la tarea de convencer al juez de que este es un ser humano de un sexo determinado. Si una vez terminada la fase y las preguntas el juez no logra determinar cuál es el humano y cuál es el Chatbot entonces la inteligencia artificial habrá superado el test de Turing.

Dado que el A.L.I.C.E. AIML fue liberado bajo una Licencia de Publico General y la mayoría de los interpretes de AIML son de licencia gratuita, muchos “clones” de ALICE han sido creados en base a las implementaciones originales del programa. El AIML gratis puede encajar en cualquier lenguaje de programación y ha sido desarrollado y adaptado a los distintos lenguajes por la misma comunidad de usuarios.



Figura 7 Logotipo de A.L.I.C.E.

2.9.1.2 FUNCIONAMIENTO

“AIML describe un grupo de objetos de datos conocidos como objetos AIML y describe la conducta con la que manejará estos objetos. Esta conducta es el cuerpo del sistema de chat entero.” (Wei, 2016).

El funcionamiento del lenguaje AIML está basado 3 etiquetas (tags) principales llamados Categorías, Patrones y Plantillas y 2 etiquetas opcionales Tópicos y Thats. También está la etiqueta que abarca todo el programa llamada **AIML**.

La etiqueta **AIML (<aiml>)** marca el inicio y el final del documento AIML. Contiene la versión y los atributos de codificación.

```
<aiml version = "1.0.1" encoding = "UTF-8"?>
...
</aiml>
```

Las **Categorías (<category>)** son la unidad básica y fundamental de conocimiento de AIML, una categoría consiste de dos elementos más: el **Patrón** y la **Plantilla**, en donde el chatbot comparará el contenido del Patrón con la pregunta que realice el usuario y si estas coinciden le entregará la respuesta que contiene la Plantilla asignada a esa categoría.

Los **Patrones (<pattern>)** son una serie de caracteres designada para ser comparada con los datos ingresados por el usuario, esta puede ser literal como por ejemplo:

“¿Cuál es tu nombre?”, o puede tener un comodín para aceptar cualquier palabra en la frase: “¿Cuál es tu *?”, donde * puede ser cualquier cadena de caracteres como tamaño, color, nacionalidad, etc.

Las **Plantillas (<template>)** son las respuestas específicas del Chatbot para patrones igualados correctamente y puede utilizar variables o funciones para dar esta respuesta.

```
<category>
  <pattern>CUAL ES TU NOMBRE</pattern>
  <template>Mi nombre es Miguel Córdoba.</template>
</category>
```

En este ejemplo tenemos una categoría de AIML en donde el Patrón es una pregunta para saber el nombre, si la entrada de datos proporcionada por el usuario es igualada correctamente con este Patrón, entonces el Chatbot dará la respuesta puesta en la Plantilla entregando así la información solicitada.

Tag de AIML	Descripción	Sintaxis	Resultado
<aiml>	Este tag define el archivo xml como un AIML. Debe estar al inicio y al final del archivo.	<aiml version="2.0"> <category> <pattern></pattern> <template></template> ></category> </aiml>	
<category>	Tag para definir una categoría de AIML, la unidad fundamental de conocimiento. Contiene <pattern> y <template>.	<category> <pattern>hola</pattern> <template>holamundo</template> ></category>	Entrada: hola Salida: holamundo
<pattern>	Tag para definir la cadena de caracteres con la que se comparará la entrada del usuario.	<pattern>HOLA</pattern>	Entrada: hola
<template>	Tag para definir la respuesta al <pattern> asociado.	<template>Holamundo </template>	Salida: holamundo
<that>	Este tag se usa para que el aiml de una respuesta en base a un contexto específico. La respuesta anterior del bot será especificada con <that> y será requerida junto con el <pattern> para que responda al template específico	<pattern>como estas</pattern> <that>holamundo</that> <template>bien gracias</template>	Entrada: hola Salida: holamundo Entrada: ¿cómo estás? Salida: bien gracias
<star>	Este tag se le asigna a una variable wildcard dentro del tag <pattern>. "n" señala la posición de * dentro del <pattern>	<pattern>El * es muy *</pattern> <template> el <star index = "1"> no es tan <star index = "2"> como crees </template>	Entrada: El árbol es muy alto Salida: el árbol no estan alto como crees
<set>	Tag que se usa para asignar un valor a una variable específica.	<pattern>hola soy *</pattern><template> hola<set name = "usuario"> <star/>! </set></template>	Entrada: hola soy botmaster Salida: hola botmaster!
<get>	Tag usado para obtener el nombre asignado previamente a una variable.	<pattern>adios</pattern> <template> hasta luego<get name = "usuario"> </template>	Entrada: adios Salida: hasta luego botmaster

Tabla 1 Tags básicos de AIML 1.0

El **Tópico (<topic>)** es una etiqueta usada en AIML para almacenar el contexto de una conversación para que posteriormente se pueda dar una conversación basada en ese contexto. Generalmente esta etiqueta se usa en conversaciones de tipo Si/No. Ayuda a AIML a buscar categorías escritas dentro del contexto del tópico.

La etiqueta **That (<that>)** se utiliza para referirse a lo que dijo el bot en una conversación anterior a una entrada del usuario. Usado mayormente en preguntas del tipo de respuesta Si/No.

La etiqueta **Srai (<srai>)** es una de las fundamentales en el funcionamiento del AIML ya que es capaz de cumplir distintas funciones. Cuando el interprete encuentra una etiqueta <srai> en una plantilla lo que hace es normalizar el contenido y volver a compararlo con patrones de otras categorías, generando una acción recursiva en la respuesta.

El uso más común de esta etiqueta es “<srai><star/></srai>” el cual se ha abreviado como <sr/>.

Step	normalized input	matching pattern	template	response
1.	PUEDES DECIR ESO DENUEVO ALICE	_ <bot name="name"/>	<sr/>	
2.	PUEDES DECIR ESO DENUEVO	_ DENUEVO	¿Otra vez? <sr/>	¿Otra vez?
3.	PUEDES DECIR ESO	PUEDES *	<sr/>	¿Otra vez?
4.	DECIR ESO	DECIR *	"<person/>"	¿Otra vez? "eso".

Tabla 2 Acción recursiva de SRAI

<srai>	Tag que se usa para hacer referencia a un <pattern> desde un determinado <template> para generar una recursion.	<pattern>saludo</pattern> <template>holamundo </template> <pattern>buenos dias</pattern> <template><srai>saludo </srai></template>	Entrada: buenos dias Salida: holamundo
<input>	Este tag replica la entrada completa del usuario que ha sido asociada con el <pattern>	<category> <pattern>deja de repetirme</pattern> <template><input /></template> </category>	Entrada: deja de repetirme Salida: deja de repetirme
<topic>	Este tag al igual que <that> se usa para contextualizar las categorias del bot según el nombre que tenga este. El valor de este tag se le asigna a la variable "topic".	<topic name="cafe"> <category> <pattern>yo lo bebo sin agregados</pattern> <template>yo prefiero el mio con azúcar y crema</template> </category> </topic>	Entrada: yo lo bebo sin agregados Salida: yo prefiero el mio con azúcar y crema
<think>	Este tag permite asignar variables sin tener que ser mostradas al usuario por medio de texto.	<category> <pattern>mi nombre es * </pattern> <template> <think><set name="nombre"><star /></set></think> Recordaré tu nombre </template> </category>	Entrada: mi nombre es juan Salida: Recordaré tu nombre

Tabla 3 Tags complejos de AIML 1.0

Etiqueta **<condition>** es una manera de general respuestas condicionadas por parte del AIML, el uso de esta puede variar dependiendo de su sintaxis pero la más usada es generando una variable que determinará cada una de las respuestas.

<condition>	Este tag se utiliza para crear una función IF THEN ELSE. Verifica un valor para determinar cual respuesta dar. Si no se asocia con ninguno entonces elige la opción por defecto.	<pre> <category> <pattern>el día esta helado</pattern> <template> <condition name="temporada"> <li value="invierno">Es debido al invierno y las lluvias <li value="verano">Pronto vendrá el calor del verano.. Yo tambien pienso lo mismo. </condition> </template> </category> </pre>	Entrada: el día esta helado Salida: Es debido al invierno y las lluvias
<random>	Este tag hace una elección aleatoria entre la lista de respuestas que tenga una determinada categoría.	<pre> <category> <pattern>*</pattern> <template> <random> ¿Qué fue lo que dijiste? No lo entiendo. ¿Puedes decirlo de otra forma? </random> </template> </category> </pre>	Entrada: #?@qwert Salida: No lo entiendo.

Tabla 4 Tags condicionales de AIML 1.0

“La base de datos es un punto crucial en la conversión automática de Voz A Texto (STT). Para cualquier sistema de reconocimiento de voz el primer paso es configurar la base de datos.” (Ghadage, 2016).

La etiqueta **<learn>** y **<learnf>** son la nueva implementación de AIML 2.0. Estos tags permiten añadir conocimientos nuevos al Chatbot mientras este mantiene una conversación.

La función del tag **<learn>** es agregar una nueva categoría al conocimiento del bot para que este pueda maximizar su asertividad y rango de respuestas correctas con el usuario. El tag **<learnf>** cumple la misma función pero guarda la nueva categoría en un archivo para que pueda ser utilizada en todas las sesiones futuras.

Tag de AIML	Descripción	Sintaxis	Resultado
<learn>	Este es el principal tag de AIML 2.0 que permite aprender nuevas categorías.	<pre> <category> <pattern>MI NOMBRE ES *</pattern> <template> <think> <set name="myname"><star /></set> </think> <learn> <category> <pattern>CUAL ES MI NOMBRE</pattern> <template>TU NOMBRE ES <eval><get name="myname" /></eval></template> </category> </learn> OK, recordaré tu nombre <get name="myname" /> </template> </category> </pre>	Entrada: Mi nombre es Pedro Salida: OK, recordate tu nombre Pedro
<learnf>	Este tag es igual a <learn> pero permite guardar los datos para todas las sesiones posteriores.	<pre> <category> <pattern>MI RUT ES *</pattern> <template> <think> <set name="myrut"><star /></set> </think> <learn> <category> <pattern>CUAL ES MI RUT</pattern> <template>TU RUT ES <eval><get name="myrut" /></eval></template> </category> </learn> OK, recordaré que tu rut es <get name="myrut" /> </template> </category> </pre>	Entrada: mi rut es 18925918-5 Salida: OK, recordaré que tu rut es 18925918-5
<vocabulary>	Tag para mostrar la cantidad de datos memorizados por el AIML	<pre> <category> <pattern>QUE TAN GRANDE ES TU VOCABULARIO</pattern> <template>Puedo reconocer <vocabulary/> palabras.</template> </category> </pre>	Entrada: ¿Que tan grande es tu vocabulario? Salida: Puedo reconocer 990 palabras.

Tabla 5 Tags incorporados en AIML 2.0

2.9.1.3 ALGORITMO DE EMPAREJAMIENTO DE PATRONES

“Uno de los métodos usados en esta aplicación es el de emparejamiento de patrones. El Chatbot emparejara la sentencia ingresada por el hablante o usuario con un patrón que existe en la base de conocimientos. (Setiaji, 2016).

A cada entrada de datos se le aplica una normalización, removiendo todas las puntuaciones, se parte en dos o más oraciones y se convierte todo a mayúsculas.

Ejemplo:

Entrada: Oye, me vas a liberar? Convertido: OYE ME VAS A LIBERAR

Luego el intérprete AIML intenta igualar palabra por palabra el emparejamiento más largo haciendo uso de caracteres especiales en caso de ser necesario para poder lograr el emparejamiento.

Los patrones de búsqueda en las categorías de AIML pueden contener wildcards de aiml, las cuales ayudan u optimizan su uso. Estas son:

- Prioridad. Empareja la palabra exacta que esté precedida por \$
- Empareja 0 o más palabras. Puede ser # o ^ según la preferencia del orden.
- Empareja 1 o más palabras. Puede ser * o _ según la preferencia del orden.
- Sets. Estos emparejan una palabra si está incluida en los sets que son filas de palabras incluidas en archivos del bot.

Los Sets, Maps, RDF y Substitution son usados por el AIML para agrupar las palabras, ya que tanto los archivos Sets, Maps y substitution contienen valores asociados a los tags. Los archivos RDF funcionan para crear una secuencia triple de Sujeto -> Predicado -> Objeto

El emparejamiento de patrones usa la técnica de búsqueda en profundidad utilizando una estructura de árbol con nodos, que contiene todos los patrones y plantillas. El uso de Sets, Maps, RDF y Substitutions puede acelerar significativamente la velocidad con la que el bot procesa una búsqueda.

2.9.1.4 GRAPHMASTER

La estructura contenedora de los archivos AIML se conoce como Graphmaster la cual consta de una colección de nodos llamados Nodemappers. Estos Nodemappers controlan las ramas de cada nodo, la cuales son palabras simples o símbolos (wildcards).

El intérprete de AIML usa un parser para construir un parse tree, en este caso llamado Graphmaster, leyendo los archivos AIML y construyendo un Pattern Path para cada categoría. Al final de cada ruta el Graphmaster contiene el enlace a la plantilla de AIML asociada con la categoría.

“Este comportamiento puede ser descrito con el conjunto de archivos y directorios en el Graphmaster, estos contienen un conjunto de nodos llamado Nodemaster y ramas que representan las primeras palabras de todos los patrones y símbolos Wildcard.” (Satu, 2015).

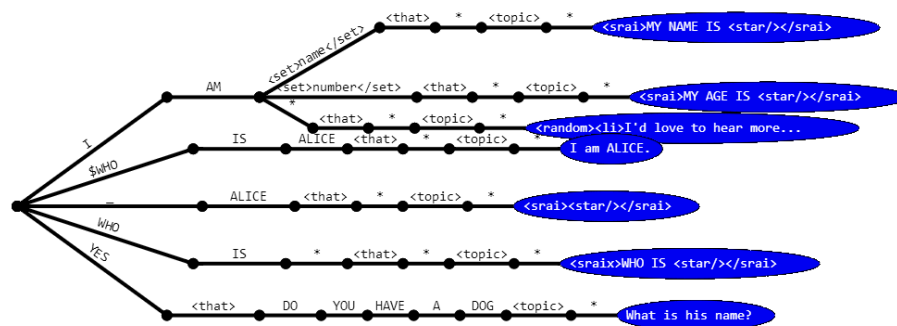


Figura 8 Graphmaster de 5 categorías

2.9.2 GOOGLE CLOUD SPEECH RECOGNITION

Google Cloud es una plataforma de Google que reúne todas las aplicaciones de desarrollo web que Google ofrece por separado. Google Cloud es el espacio virtual en el que se realizan una serie de tareas de acceso, almacenamiento y gestión de datos.

La API de Speech Recognition disponible en Google Cloud es una potente herramienta de uso en línea que permite convertir texto en audio aplicando avanzados modelos de redes neuronales, tiene soporte para más de 80 idiomas, puede transcribir el audio dictado a un micrófono, habilitar el control por voz o transcribir archivos de audio.

“El Reconocimiento de Voz Automático (ASR) es el proceso de capturar una señal acústica por medio de un micrófono u otro dispositivo de grabación, y luego transformar la señal a un texto legible.” (Prakoso, 2016).

El uso de Google Cloud Speech API es de vital importancia en el desarrollo de este proyecto, puesto que la fiabilidad de la transcripción texto-audio es una de las prioridades a la hora de una conversación con un Chatbot. Esto debido a que el AIML debe recibir el mensaje correctamente transcrito, de lo contrario podría interpretar mal la petición y dar una respuesta errónea o simplemente no encontrar la respuesta adecuada.



Figura 9 Funcionamiento de Google Cloud Speech API

2.9.3 RASPBERRY PI 3 MODEL B

Esta microcomputadora de bajo coste resulta más eficiente de utilizar para tareas específicas que un ordenador personal, ya que dada su fisionomía de hardware reducida, su consumo de recursos es muy bajo en comparación con el de un PC, por lo que es posible mantener a la Raspberry Pi operativa durante tiempo indefinido. Sin embargo, no posee grandes capacidades de memoria ni espacio de almacenamiento ni memoria de vídeo como para realizar exactamente las mismas funciones que una computadora personal, aunque algunas de estas capacidades o funcionalidades pueden ser añadidas mediante hardware externo con el que este mismo microcontrolador es compatible, por lo que es muy personalizable dependiendo de la aplicación que se le quiera dar.

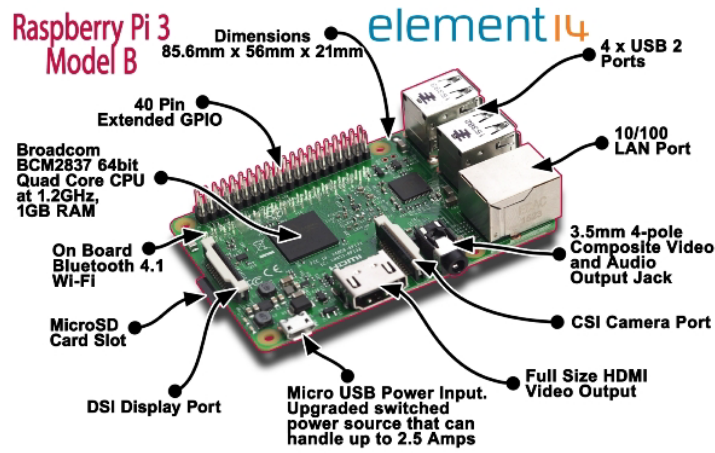






















Figura 10 Detalles Raspberry Pi 3 Model B

Raspberry Pi 3 GPIO Header					
Pin#	NAME		NAME	Pin#	
01	3.3v DC Power		DC Power 5v	02	
03	GPIO02 (SDA1 , I²C)		DC Power 5v	04	
05	GPIO03 (SCL1 , I²C)		Ground	06	
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08	
09	Ground		(RXD0) GPIO15	10	
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12	
13	GPIO27 (GPIO_GEN2)		Ground	14	
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16	
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18	
19	GPIO10 (SPI_MOSI)		Ground	20	
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22	
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24	
25	Ground		(SPI_CE1_N) GPIO07	26	
27	ID_SD (I²C ID EEPROM)		(I²C ID EEPROM) ID_SC	28	
29	GPIO05		Ground	30	
31	GPIO06		GPIO12	32	
33	GPIO13		Ground	34	
35	GPIO19		GPIO16	36	
37	GPIO26		GPIO20	38	
39	Ground		GPIO21	40	

Rev. 2
29/02/2016

www.element14.com/RaspberryPi

Figura 11 Pinout Raspberry Pi 3 Model B

2.9.4 RASPBIAN S.O.

Raspbian es un sistema operativo gratis basado en Debian (un sistema GNU basado en software libre) optimizado para la Raspberry PI. Este sistema operativo cuenta con el conjunto de programas y utilidades básicas para que la Raspberry PI funcione correctamente.

El compilado inicial consta de alrededor de 35000 paquetes de Raspbian, optimizados para funcionar en la Raspberry PI y se completó en Junio de 2012. Aun así, Raspbian aún sigue bajo desarrollo, con énfasis en optimizar la funcionalidad y estabilidad lo máximo posible.

2.9.5 FESTIVAL TTS

El software Festival Text To Speech es un programa gratis desarrollado originalmente por Alan W. Black en el Centro de Investigación de Voz (CSTR) en la Universidad de Edimburgo para la conversión de texto en audio por medio de voz. FestivalTTS es capaz de funcionar en múltiples plataformas y viene con soporte para voces inglesas (británicas y americanas), galesas y español.

Adicionalmente se pueden descargar paquetes de voces para proporcionar a Festival la capacidad de sintetizar en más idiomas.

Festival posee su propio set de herramientas llamado Festvox, el cual posee todas las herramientas para crear voces sintéticas, es gracias a este software que la capacidad multilingüe de Festival se ha expandido de gran manera.

Este software se utilizara para darle voz al AIML instalado en la Raspberry, de esta forma podrá dar una respuesta por voz satisfactoria con respecto a la entrada que el usuario a dictado al micrófono como pregunta para el Chatbot.

2.9.6 ZX-THERMOMETER

El circuito integrado de ZX-Thermometer esta compuesto por un termistor, juegos de resistencias y un amplificador operacional MCP6002. Este sensor de temperatura es conectado a 5V, GND y la señal de salida que proporciona el valor analógico de la temperatura.

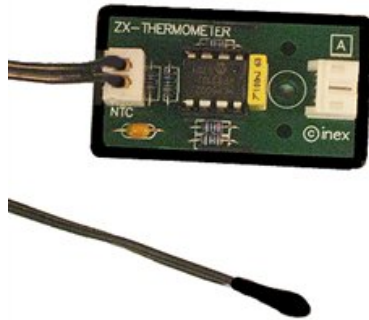


Figura 12 ZX-Thermometer sensor de Temperatura

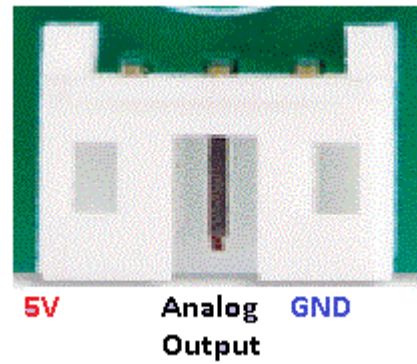


Figura 13 Pinout de ZX-Thermometer

2.9.7 I2C ADS1115

El I2C ads1115 es un conversor analógico a digital de 16 bits de resolución el cual se usa para medir la señal analógica del sensor de temperatura y transformarla para ser leída por la Raspberry Pi 3. Este conversor requiere la conexión de 3.3V, GND y los GPIO 2 (SDA) y 3 (SCL), además de esto las entradas analógicas van en los 4 canales que dispone para leer este tipo de datos

Este controlador posee una librería en Python3 por lo que su uso es simple y permite una lectura de datos rápida y eficiente dentro de lo necesario para el funcionamiento del bot.

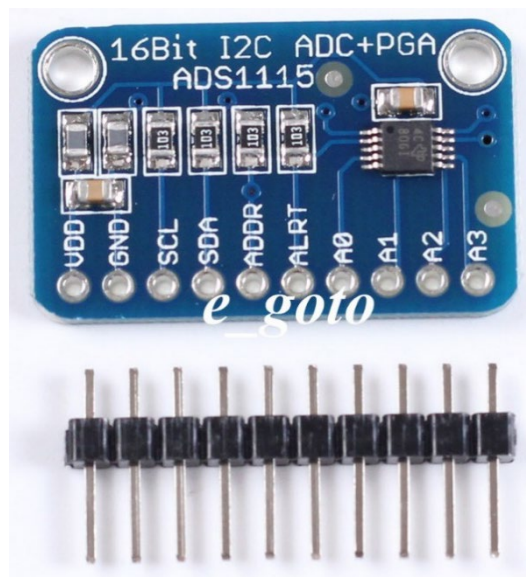


Figura 14 Conversor Analogo Digital de 16 bits de resolución ads1115

2.10 CARTA GANTT

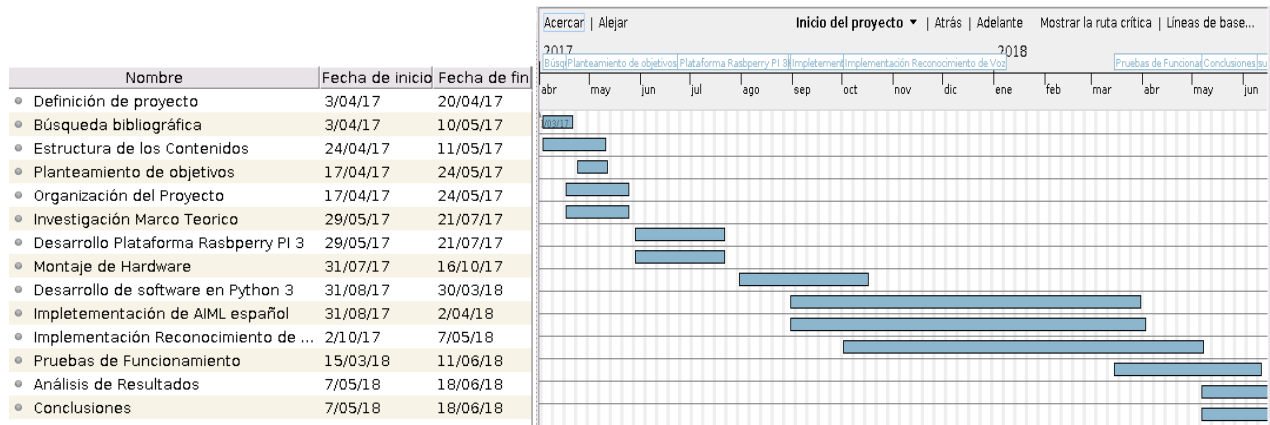


Figura 15 Carta Gantt

3 DESARROLLO DEL PROTOTIPO

3.1 DIAGRAMA DE BLOQUES

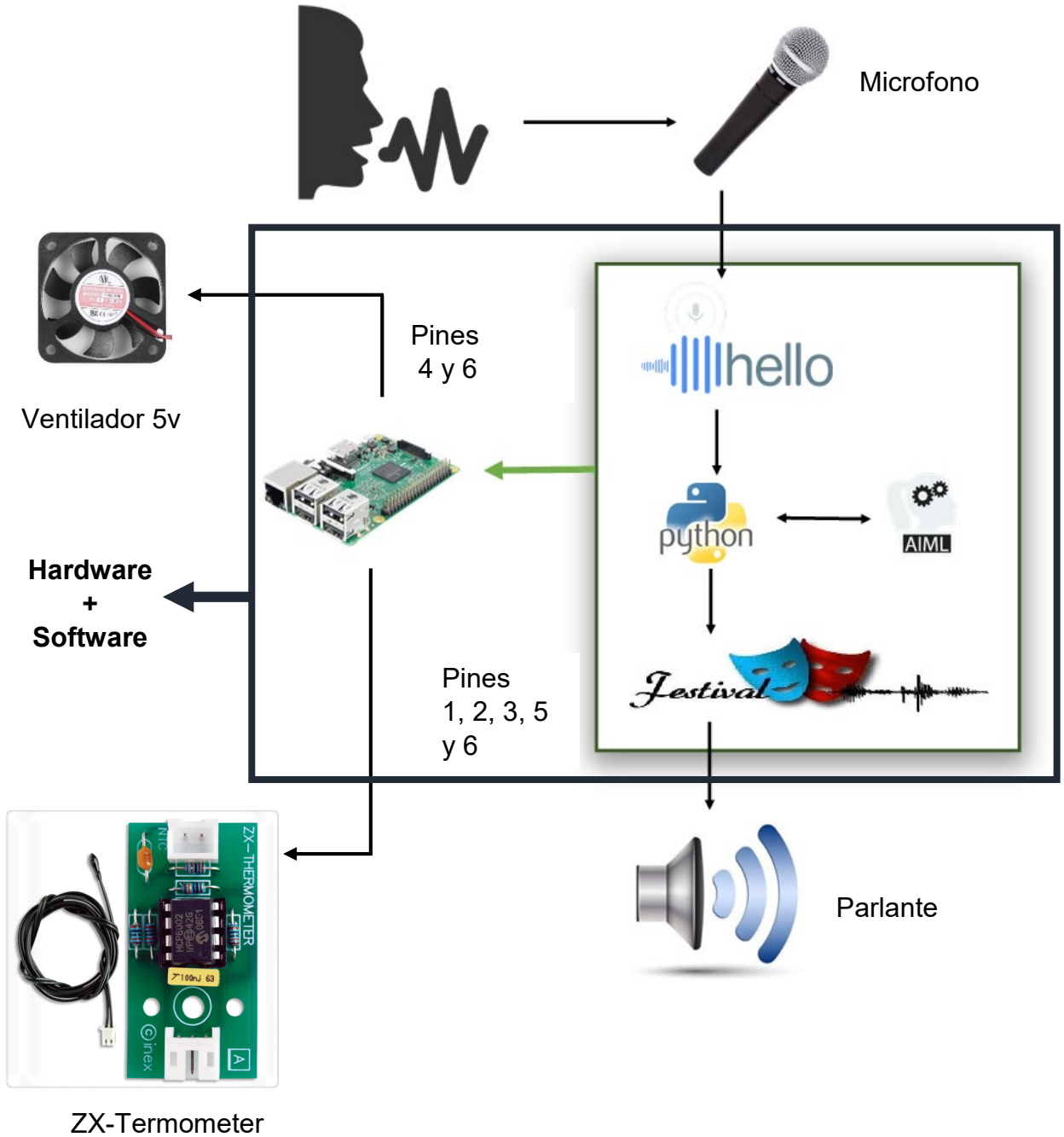


Figura 16 Diagrama de Bloques

El inicio del funcionamiento del proyecto se realiza por medio de la voz humana del usuario la cual pedirá una respuesta específica por parte de nuestra interfaz. Este mensaje es recibido por el micrófono y analizado por Google Cloud Speech API para realizar la decodificación de este en un carácter de texto legible por el software AIML de Python inserto en la plataforma. El paso siguiente es el procesamiento de AIML para retornar una respuesta satisfactoria por medio del software de Python la cual será enviada haciendo uso de Festival TTS para obtener una salida por voz hacia el usuario.

3.2 DIAGRAMA DE FLUJO

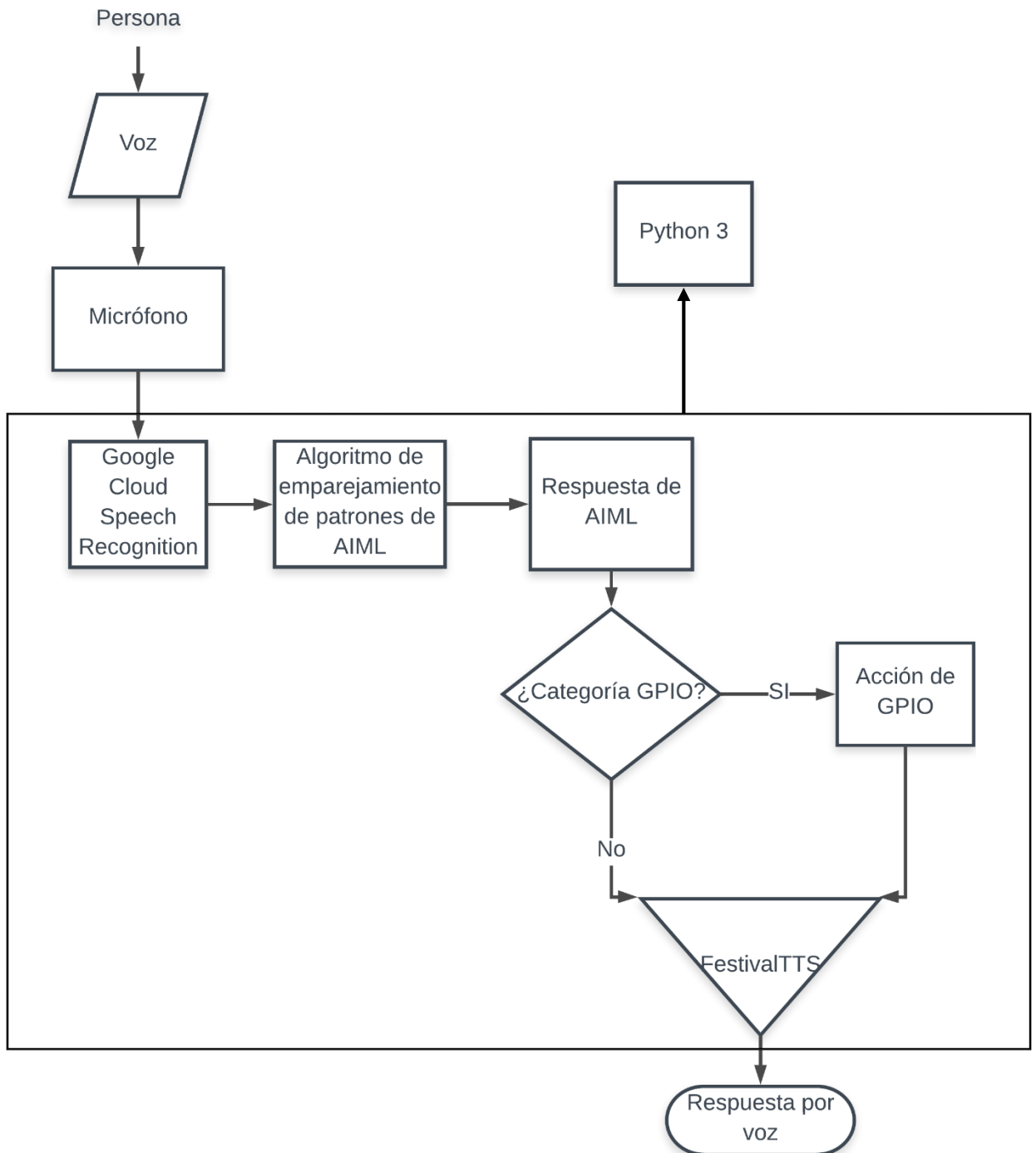


Figura 17 Diagrama de Flujo

3.3 PROCESAMIENTO DE LA ENTRADA Y SALIDA DE AUDIO

El procesamiento de la Raspberry PI 3 comienza con la entrada de audio que provendrá del usuario, este será captado a través de un micrófono conectado a una tarjeta de sonido USB colocada en el microcontrolador. Este audio será interpretado por la aplicación Google Cloud Speech API mediante Python 3, el cual lo transformará a texto, para luego ser procesado por el AIML. Cuando AIML entrega su respectiva respuesta esta es procesada por el motor Festival TTS, el cual transformara la respuesta de texto a audio para darle la voz de salida al Chatbot.

3.4 PROCESAMIENTO EN AIML

Para iniciar una conversación con el Chatbot es necesario ejecutarlo desde un archivo bash, este iniciará el cerebro del AIML, cargando todas sus categorías en conjunto con el usuario actual y los distintos sets por defecto. Este dará un saludo inicial para indicar que ya ha cargado todo y está listo para entablar una conversación.

Una vez que el AIML ha cargado este genera un parse tree (Graphmaster) el cual contiene todas las categorías dentro del conocimiento de este. Cuando el usuario ingresa una entrada el software intenta asociarla dentro de su árbol de parseo emparejando nodo por nodo las palabras hasta llegar al ultimo eslabon. El camino de los nodos seguidos por el intérprete de AIML se conoce como Pattern Path y es el recorrido que el software debe hacer para encontrar la respuesta a una entrada específica.



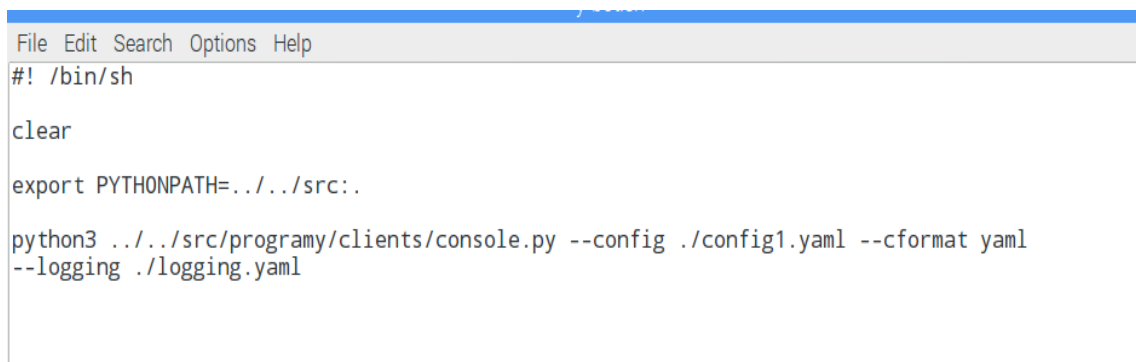
Figura 18 Pattern Path incluyendo <that> y <topic>

Si la respuesta proveniente del Chatbot no es satisfactoria o no está dentro de sus conocimientos este tendrá la capacidad de agregarlos por medio del archivo AIML badanswer.aiml. Las categorías provenientes de este archivo entran en juego cuando al bot se le indica que su respuesta anterior fue insatisfactoria, a lo cual este responderá preguntando por una respuesta correcta para agregar a su conocimiento.

Si la plantilla de respuesta en badanswer.aiml contiene el tag <learn> o <learnf> se creará una nueva categoría conteniendo la información indicada por la plantilla activada, generando así una respuesta para la pregunta que anteriormente no fue respondida.

3.5 CONFIGURACIÓN DEL BOT

La configuración del Chatbot es un archivo de texto que se encuentra en la carpeta de archivos del bot y es llamada al ser ejecutado el archivo bash. Dentro de este archivo también se especifica la consola que vamos a usar para interactuar con el bot, en este caso el básico es console.py el PYTHONPATH en la carpeta del bot y el logging.yaml para funciones específicas de Python 3.



```
File Edit Search Options Help
#!/bin/sh

clear

export PYTHONPATH=../../src:.

python3 ../../src/programy/clients/console.py --config ./config1.yaml --cformat yaml
--logging ./logging.yaml
```

Figura 19 Ejecutable .bash del bot

El archivo config puede ser .yaml, .json o .xml. Los valores contenidos dentro del archivo son los mismos independientes del formato.

Dentro del archivo config usualmente se encuentran referencias a una variable \$BOT_ROOT. Este valor reemplaza la ubicación actual del bot en la ruta permitiendo una configuración más simple en los archivos y directorios.

3.5.1 SECCION DEL BOT

Esta parte de la configuración determina las características generales del comportamiento del bot y de la forma de manejar las consultas.

Configuración Base			
Nombre	Descripción	Ejemplo	Por Defecto
prompt	Símbolo que se muestra en la consola para indicar que se ponga una sentencia	>>>	>>>
initial_question	Pregunta inicial que hace el chatbot cuando inicia	¿Qué puedo hacer por ti?	Hello
default_response	Si el chatbot no encuentra ninguna respuesta devolverá default_response por defecto.	Lo siento, no tengo una respuesta para eso	Empty String
empty_string_sra	Alternativamente a default_response, si el bot devuelve un string vacío este hará referencia a PROGRAM_EMPTY	PROGRAM_EMPTY	None
exit_response	Lo que el chatbot dice cuando se cierra la conversación.	¡Hasta luego!	Bye!
override_predicates	Los predicados son valores iniciales que se cargan al inicio. Si esta opción esta activa entonces estos pueden ser sobrescritos por los archivos de gramática.	True	False

Tabla 6 Configuración de bot 1

Configuración Base			
Nombre	Descripción	Ejemplo	Por Defecto
max_question_recursion	Si la misma pregunta se repite muchas veces con un tag <srai> el bot puede quedar en un loop infinito. Esta es la cantidad de veces máximas que puede generar una respuesta recursiva.	100	100
max_question_timeout	Alternativo a max_question_recursion, este tag pone un límite de tiempo para descartar la pregunta. El valor por defecto -1 es para que no haya timeout	1	-1
max_search_depth	El máximo de profundidad para ejecutarlas las búsquedas dentro del parse tree	66	100
max_search_timeout	Alternativo a max_search_depth pero define el tiempo máximo que puede esperar en la búsqueda de una respuesta	1	-1

Tabla 7 Configuración de bot 2

3.5.2 CONFIGURACIÓN DE ORTOGRAFÍA

La configuración de ortografía se usar para determinar la clase que el bot usará para revisar ortográficamente cada sentencia. El revisador ortográfico se apoya en un corpus de palabras para revisar cada oración, mientras más grande el corpus más tiempo toma pero también es más efectivo al chequear.

Hay dos formas para controlar la manera en la que se realiza la revisión ortográfica, chequear cada sentencia antes de hacer la pregunta o chequear solamente si la pregunta falla al encontrar respuesta y reintentar con la pregunta corregida.

Nombre	Descripción	Ejemplo	Por Defecto
classname	Ruta completa a la clase conteniendo el Spelling de Python	programy.utils.spelling.checker.SpellingChecker	None
corpus	Ruta completa hasta el corpus de texto	\$BOT_ROOT/spelling/corpus.txt	None
check_before	Chequear ortografía antes de hacer la pregunta	true	false
check_and_retry	Solo chequear si la pregunta falla al encontrar respuesta.	true	false

Tabla 8 Configuración de ortografía

3.5.3 CONFIGURACIÓN DEL CEREBRO DEL BOT

La configuración del “cerebro” del bot controla como este interpreta las preguntas ingresadas y como entrega las respuestas. Este cerebro posee una gran cantidad de opciones configurables, la mayoría con valores por defecto por lo que no necesita mayor configuración para su primer uso.

Configuración	Descripción	Funcionalidad
Overrides	Sección que corresponde a la configuración de las funciones habilitadas del bot	Habilitar funcionamiento del tag learn, learnf y system
Defaults	Sección donde se definen los valores por defecto de las variables usadas por el bot	Definir valor por defecto para get, property, map y directorio de learnf.aiml
Nodos	Configuración de la ruta en la que se guardan los archivos correspondientes a la formación de los nodos	Definir la ruta completa al directorio de la representación de los nodos de patrones y plantillas
Binarios	Opciones para habilitar la generación de un respaldo binario del cerebro	Funciones de guardado y carga de cerebro binario
Archivos AIML	Opciones para definir la ruta en donde se encuentran los archivos aiml y donde guardar los logs del bot	Definición de directorio, recursion y extensión de archivos AIML y logs de errores, duplicado y conversaciones
Archivos SET	Configuración del directorio, extensión y recursion de los archivos SET	Archivos que agrupan varios nombres en distintos sets para ser usados por el bot con el tag <set name>
Archivos MAP	Configuración del directorio, extensión y recursion de los archivos MAP	Archivos que determinan una relación de una frase o palabra para que el bot responda con otra con el tag <map name>
Archivos RDF	Configuración del directorio, extensión y recursion de los archivos RDF	Archivos que guardan un conjunto de 3 elementos Sujeto-Predicato-Objeto.
Archivos Substitution	Configuración del directorio, extensión y recursion de los archivos Substitution	Archivos que definen una sustitución de una palabra en la respuesta del chatbot por otra asociada al tag de la sustitución
Archivos Processor	Configuración del directorio, extensión y recursion de los archivos Processor	Procesamiento de la entrada antes de ser ingresada el cerebro del bot y procesamiento de la salida antes de ser entregada al usuario

Tabla 9 Configuraciones del cerebro

3.5.3.1 OVERRIDES DEL CEREBRO

Configuraciones para permitir o denegar tags específicos del funcionamiento del Chatbot.

Nombre	Descripción	Ejemplo	Por Defecto
allow_system_aiml	Permite el uso del tag <system> que otorga información sobre el sistema operativo actual.	[True False]	True
allow_learn_aiml	Permite el uso del tag <learn> para que el bot aprenda nuevas categorías.	[True False]	True
allow_learnf_aiml	Permite el uso del tag <learnf> para aprender nuevas categorías pero que además persistan el reinicio del bot.	[True False]	True

Tabla 10 Configuración de overrides

3.5.3.2 DEFAULTS DEL CEREBRO

Configuraciones para definir el valor por defecto de las variables que posee el Chatbot.

Nombre	Descripción	Ejemplo	Por Defecto
default-get	Si un tag <get> no tiene nombre retorna este valor.	Nada	Unknown
default-property	Si un predicado no tiene valor, retorna este valor.	Nada	Unknown
default-map	Si un map no tiene valor asignado, devuelve esto.	Nada	Unknown
learn-filename	Ruta completa hacia el archivo que guardara las categorías aprendidas con <learnf>	\$BOT_ROOT/learnf.aiml	learnf.aiml

Tabla 11 Configuración de defaults

3.5.3.3 NODOS DEL CEREBRO

Configuración de la ruta de almacenamiento de archivos del procesamiento de los nodos.

Nombre	Descripción	Ejemplo	Por Defecto
pattern_nodes	La ruta completa al archivo de configuración que contiene el mapeo de los nodos pattern de python	<code>\$BOT_ROOT/config/pattern_nodes.conf</code>	None - Archivo de mapeo interno usado
template_nodes	La ruta completa al archivo de configuración que contiene el mapeo de los nodos template de python	<code>\$BOT_ROOT/config/template_nodes.conf</code>	None - Archivo de mapeo interno usado

Tabla 12 Configuración de nodos

3.5.3.4 BINARIOS DEL CEREBRO

Configuraciones para guardar un respaldo del cerebro y disminuir su tiempo de carga a costa de algunas funcionalidades.

Nombre	Descripción	Ejemplo	Por Defecto
save_binary	Guarda la versión binaria del cerebro en un archivo para cargarla después. Esto acelera mucho la velocidad de carga del cerebro, pero debe ser usado con cautela ya que no guarda los datos aprendidos con <learnf>	[True False]	False
load_binary	Carga una versión previamente guardada del cerebro binario. Útil en producción una vez que el cerebro está completo, pero no guarda las categorías aprendidas con <learnf>	[True False]	False
binary_filename	Nombre del archivo binario para guardar y cargar.	\$BOT_ROOT/brain s/p roduction.brain	None
load_aiml_on_bin ary_fail	Si el cerebro falla al cargar esta opción le señala al bot que debe cargar los archivos AIML puros según la configuración de archivos.	[True False]	False
dump_to_file	Ruta para guardar una representación textual del parse tree. Una vez que los archivos AIML han sido cargados este árbol es traspasado a este archivo txt	\$BOT_ROOT/braint ree.txt	None

Tabla 13 Configuración de binarios

3.5.3.5 ARCHIVOS DEL CEREBRO

Todos los bots que utilizan la versión 2.0 de AIML requieren una gran cantidad de archivos para crear su cerebro. En esta sección de archivos es donde se manejan estos datos y lógicas. Hay dos maneras de configuración para los archivos.

Archivos Múltiples: Cuando la configuración requiere saber un número desconocido de archivos se especifica el directorio, la extensión del archivo y un flag que indica si se añadirán los subdirectorios.

Archivos Simples: Cuando solo se requiere un archivo para la configuración solo se necesita la ruta directa hacia este mismo.

```
files:
  aiml:
    files: $BOT_ROOT/aiml
    extension: .aiml
    directories: true
  errors:
    file: $BOT_ROOT/logs/y-bot_errors.txt
    file: $BOT_ROOT/logs/y-bot_errors.csv
    format: csv
    encoding: utf-8
    delete_on_start: true
  duplicates:
    #file: $BOT_ROOT/logs/y-bot_duplicates.txt
    file: $BOT_ROOT/logs/y-bot_duplicates.csv
    format: csv
    encoding: utf-8
    delete_on_start: true
  conversation:
    #file: $BOT_ROOT/logs/y-bot-conversation.txt
    file: $BOT_ROOT/logs/y-bot-conversation.csv
    format: csv
    delete_on_start: true
  sets:
    files: $BOT_ROOT/sets
    extension: .txt
    directories: false
  maps:
    files: $BOT_ROOT/maps
    extension: .txt
    directories: false
```

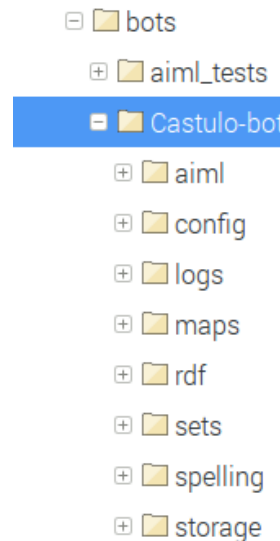


Figura 20 Configuración de archivos del bot

Figura 21 Organización de carpetas del bot

3.5.3.6 ARCHIVOS AIML

Hay dos variantes en esta configuración, la primera carga todos los aiml encontrados en la ruta especificada en 'files' y opcionalmente todos los archivos en los subdirectorios si 'directories' es verdadero.

Nombre	Descripción	Ejemplo	Por Defecto
files	Directorio que contiene los archivos AIML a cargar.	\$BOT_ROOT/aiml	.
extension	Extensión de los archivos a cargar	Aiml	aiml
directories	Buscar recursivamente y cargar todos los archivos de los subdirectorios dentro de la ruta especificada en 'directories'	true	true
errors	Ruta a un archivo en el cual escribe todos los errores cuando son encontrados.	\$BOT_ROOT/logs/errors.txt	None
duplicates	Ruta a un archivo en el cual escribe todos los duplicados que encuentra.	\$BOT_ROOT/logs/duplicates.txt	None
Conversación	Ruta a un archivo en el cual escribe todos los pares de pregunta/respuesta a medida que son respondidos.	\$BOT_ROOT/logs/conversation.txt	None

Tabla 14 Configuración de archivos aiml

La segunda opción es especificar los archivos aiml a cargar uno por uno usando el atributo 'file'. En este caso 'directories' y 'extension' son ignorados y en 'file' se debe indicar un archivo aiml valido.

file	Ruta completa a un único archivo de aiml a cargar	\$BOT_ROOT/aiml/autos.aiml	None
------	---	----------------------------	------

También hay una configuración alternativa para guardar los errores, duplicados y conversaciones, la que permite especificar el tipo de archivo y la codificación usada.

Nombre	Descripción	Ejemplo	Por Defecto
File	Ruta a archivo específico para guardar los datos	/tmp/errors.csv	None
format	Formato del archivo	[txt csv]	txt
encoding	Formato de codificación del archivo	utf-8	utf-8
delete_on_start	Borrar cuando inicia el sistema para así usar la información más nueva.	[True False]	False

Tabla 15 Configuración alternativa de errores

3.5.3.7 ARCHIVOS SET

Los archivos set son usados por el tag <set>. El nombre del archivo (sin la extensión) se convierte en el nombre del set.

```
<pattern>  
  Que es * <set name="animals" />  
</pattern>
```

Nombre	Descripción	Ejemplo	Por Defecto
files	Directorio que contiene todos los archivos map.	\$BOT_ROOT/sets	None
extension	Extensión de los archivos a cargar.	txt	None
directories	Flag para decirle al bot si revisar los subdirectorios en la carpeta definida en 'files'	[True False]	True

Tabla 16 Configuración de sets

3.5.3.8 ARCHIVOS MAP

Los archivos map son usados por el tag <map>. El nombre del archivo (sin la extensión) se convierte en el nombre del map.

```
<template>  
  Lo opuesto de <star /> es <map name="opposite" /><star/></map>  
</template>
```

Nombre	Descripción	Ejemplo	Por Defecto
files	Directorio que contiene todos los archivos set	\$BOT_ROOT/sets	None
extension	Extensión de los archivos a cargar	txt	None
directories	Flag para avisarle al bot si revisar los subdirectorios de la ruta especificada en 'files'	[True False]	True

Tabla 17 Configuración de maps

3.5.3.9 ARCHIVOS RDF

Los archivos rdf son usados por los tags <addtriple>, <deletetriples>, <select> y <uniq>. Los archivos rdf especifican Sujeto -> Predicado -> Objeto mapeado

Nombre	Descripción	Ejemplo	Por Defecto
files	Directorio que contiene todos los archivos rdf	\$BOT_ROOT/rdf	None
extension	Extensión de los archivos a cargar	txt	None
directories	Flag para avisarle al bot si revisar los subdirectorios de la ruta especificada en 'files'	[True False]	True

Tabla 18 Configuración de archivos rdf

3.5.3.10 ARCHIVOS SUBSTITUTION

Los archivos substitution se asocian a sí mismos con un tag del template. Todos estos archivos funcionan de la misma manera, reemplazando un valor encontrado en el tag por uno definido según la relación en el archivo de sustitución.

Nombre	Descripción	Ejemplo	Por Defecto
denormal	Ruta completa a la lista de valores manejados por el tag <denormal>	\$BOT_ROOT/config/denormal.txt	None
normal	Ruta completa a la lista de valores manejados por el tag <normal>	\$BOT/config/nomal.txt	None
gender	Ruta completa a la lista de valores manejados por el tag <gender>	\$BOT/config/gender.txt	None
person	Ruta completa a la lista de valores manejados por el tag <person>	\$BOT/config/person.txt	None
person2	Ruta completa a la lista de valores manejados por el tag <person2>	\$BOT/config/person2.txt	None
properties	Ruta completa a la lista de valores manejados por el tag <get>	\$BOT/config/properties.txt	None
variables	Cuando un nuevo cliente es identificado sus variables de conversación (accesible como `name='xx'`) son cargadas.	\$BOT/config/variables.txt	None

Tabla 19 Configuración de archivos substitutions

3.5.3.11 ARCHIVOS PROCESSOR

Los pre procesadores y los post procesadores son usados para transformar el texto antes de y después de que la pregunta es realizada. Estos son usados para formatear el texto a mostrar o para remover alguna información ambigua como puntuaciones.

Pre-Processor: Ejecutado en la entrada del usuario antes de ser pasado al cerebro.

Post-Processor: Ejecutado en el último paso en la respuesta del bot antes de ser mostrado al usuario.

Nombre	Descripción	Ejemplo	Por Defecto
preprocessors	Ruta completa al archivo que contiene la lista de pre procesadores definidos de Python	\$BOT_ROOT/config/preprocessors.txt	None
postprocessors	Ruta completa al archivo que contiene la lista de post procesadores definidos de Python	\$BOT_ROOT/config/postprocessors.txt	None

Tabla 20 Configuración de archivos processor

3.6 CÓDIGO EN PYTHON 3

AIML es un lenguaje del tipo Natural Language Processing el cual se basa en establecer un medio de comunicación común tanto para el humano como para la máquina. Si bien AIML posee todo el núcleo del procesamiento, este necesita un lenguaje o una plataforma que usar de intérprete para el manejo de sus datos. En este caso el intérprete disponible de AIML 2.0 a usar es uno hecho en Python 3, el cual contiene los códigos necesarios para ejecutar el Chatbot.

El Chatbot viene funcionando al 100% con su configuración por defecto, por lo que lo único que debemos agregar nosotros a su código es la parte de Reconocimiento de Voz, Text to Speech y GPIO. Para lograr un correcto funcionamiento de estos programas entre si y con Python3 es necesario descargar e instalar las librerías necesarias.

Librería	Descripción	Instalación
PyYaml	El paquete pyyaml es un parser que permite a Python3 trabajar con el lenguaje de tipo yaml	sudo pip3 install pyyaml
PyAudio	El paquete de pyaudio le da a Python3 la capacidad de poder grabar y reproducir sonidos en una variedad de plataformas	sudo pip3 install pyaudio
SpeechRecognition	Librería de reconocimiento de voz desarrollada para Python y Python3. Posee soporte para varios programas de reconocimiento, entre ellos Google Cloud Speech	sudo pip3 install SpeechRecognition
PyFestival	Paquete que provee los archivos para permitir a Python3 usar el sintetizador de voz FestivalTTS	sudo pip3 install pyfestival
python-dev y smbus	Python-dev se requiere para la compilación de Python3 y smbus para la comunicación con puertos GPIO de la Raspberry Pi 3	sudo apt-get install python-dev python-smbus
Adafruit-ads1x15	Librería del ADC ads1115 desarrollada por adafruit para ser utilizada en Python3	sudo pip3 install adafruit-ads1x15
Rpi.GPIO	Esta librería permite usar los pines GPIO de Raspberry Pi 3 mediante Python3	sudo pip3 install Rpi.GPIO

Tabla 21 Librerías necesarias para Python3

3.6.1 RECONOCIMIENTO DE VOZ

```
Import Speech_recognition as sr # Importacion de la librería Speech_recognition

GOOGLE_CLOUD_SPEECH_CREDENTIALS = r"""INSERT THE CONTENTS OF THE GOOGLE CLOUD
SPEECH JSON CREDENTIALS FILE HERE"""

With sr.Microphone() as source:

    Audio = r.listen(source)

    Audioatexto = r.recognize_google_cloud(audio,
credentials_json=GOOGLE_CLOUD_SPEECH_CREDENTIALS)
```

En la parte de Speech Recognition es necesario importar el paquete de Python3 para reconocimiento de voz llamado `speech_recognition`. Luego se deben incorporar las credenciales de Google Cloud Speech. Finalmente usando el micrófono se guarda la grabación de voz en la variable “audio” y finalmente se traspa a texto en la variable “audioatexto” utilizando la función Google Cloud Speech.

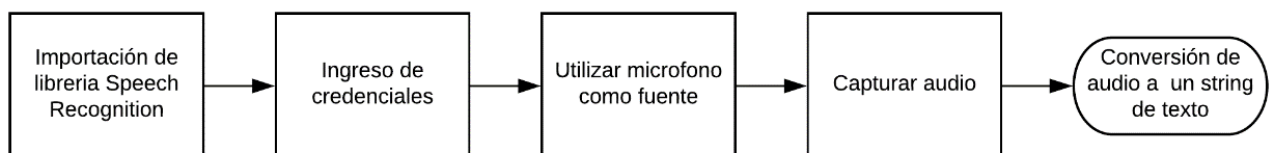


Figura 22 Diagrama del código de Reconocimiento de Voz

3.6.2 CONVERSIÓN DE TEXTO A VOZ

```
import os

texto="holamundo"
os.system('echo "{0}" |iconv -f utf-8 -t iso-8859-1| festival --tts'.format(texto))
```

Figura 23 Código de FestivalTTS

La parte de Texto to Speech es más simple que la anterior, solo se necesita descargar el paquete de FestivalTTS para Python3 (pyfestival). Una vez instalado, debemos importar el modulo “os” que nos sirve para escribir comandos de línea a través de Python3 y finalmente utilizamos la función “system” del módulo “os” para indicar lo que queremos que sea transformado. Utilizamos el comando “echo” para indicar que debe haber una salida por voz, luego las configuración de “iconv -f utf-8 iso-8859-1” hacen una transformación del string para que festival no tenga problemas con las puntuaciones o símbolos desconocidos de nuestro lenguaje, como último paso especificamos a festival que utilice su función ---tts y el texto que queremos que lea, en este caso guardado en la variable “texto”.

3.6.3 CONFIGURACIÓN DE GPIO

Para que nuestro Chatbot posea las capacidades de interactuar con el entorno se deben configurar los pines de propósito general (GPIO) de Raspberry Pi 3. Primero se deben importar las librerías de GPIO, luego establecer la configuración para usar los puertos y finalmente definir los pines a utilizar como salida en este caso. Para activar los pines utilizamos los comandos “True” (activo o 1) o “False” (inactivo o 0).

```
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)
GPIO.setup(21, GPIO.OUT)
GPIO.output(21, True)
GPIO.output(21, False)
```

Figura 24 Código para utilizar GPIO como output

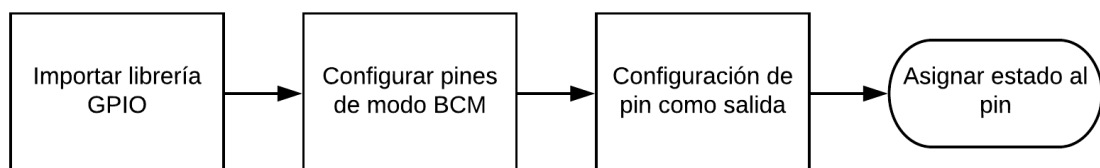


Figura 25 Diagrama del código de GPIO

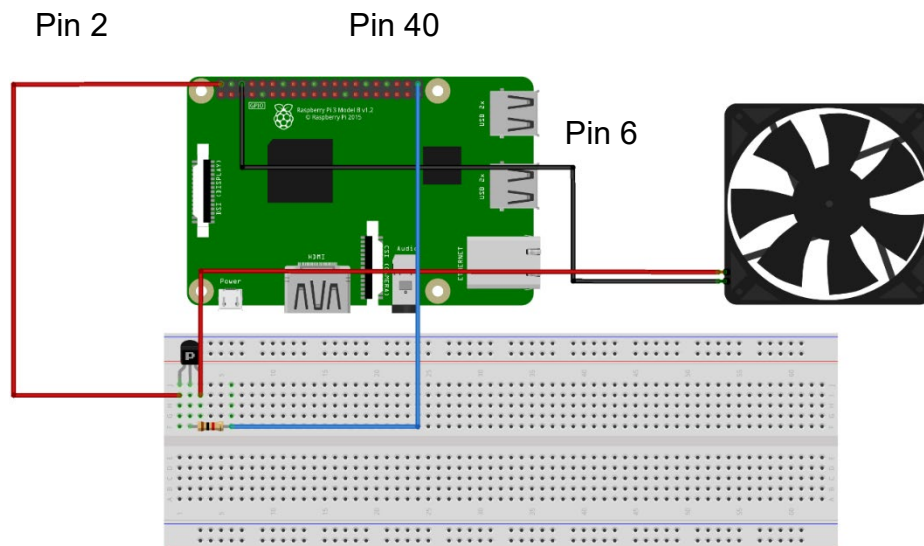


Figura 26 Conexión de ventilador 5V

El conexionado se ha realizado usando un transistor K30A FET de canal N para activar el ventilador a través de la salida GPIO 21.

```
import Adafruit_ADS1x15
adc = Adafruit_ADS1x15.ADS1115()
GAIN = 1
valortemp = adc.read_adc(0, gain=GAIN)
```

Figura 27 Configuración del ADC en Python 3

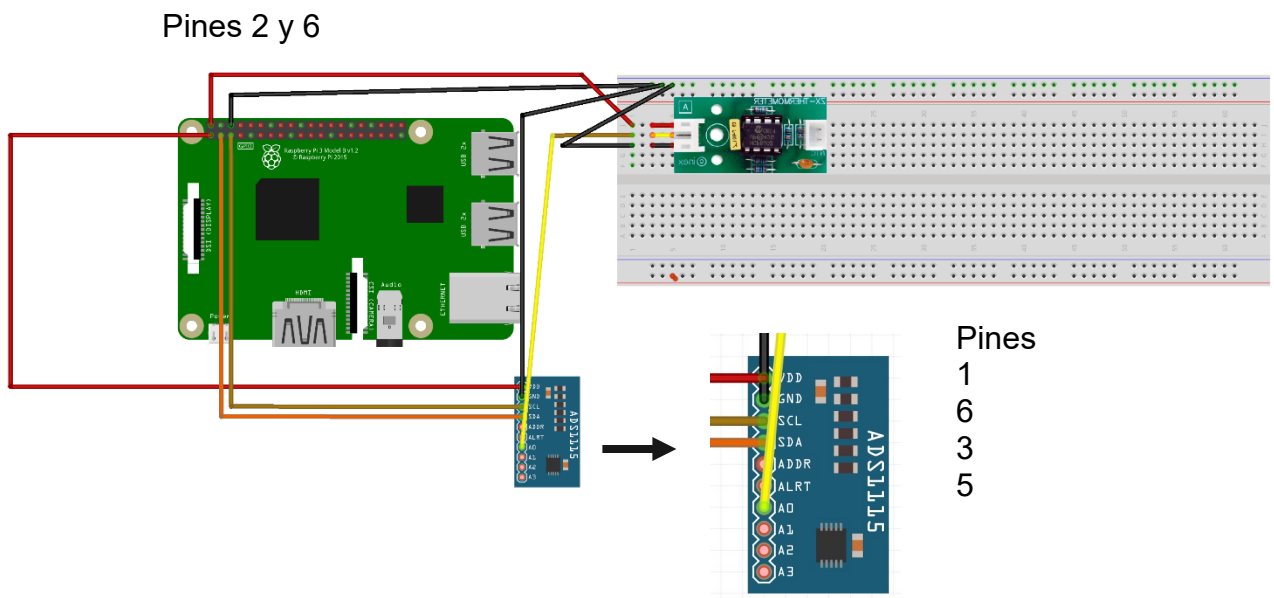


Figura 28 Conexión de I2C ads1115 y ZX-Thermometer

3.6.3 CONSOLA DEL CHATBOT

La consola del Chatbot es la interfaz principal del intérprete, es decir, es la que sirve de comunicación entre el lenguaje AIML y el usuario. Esta consola está definida dentro de un código de Python llamado `console.py`, el cual establece las características de esta y además contiene todos los llamados y funciones a realizar durante la conversación del Chatbot.

Este código de la consola hace una importación de características para definir sus parámetros de inicio. Importando `BotClient` define las características iniciales del bot, como la configuración de sus variables y del usuario en cuestión con el que está teniendo la conversación. La importación de `ConsoleConfiguration` define la ruta de los archivos a usar y también la manera en que se muestra la consola hacia el usuario.

```
import logging                                # Importación de logging para generar archivo
de log
import speech_recognition as sr # Importación del paquete de speech_recognition
import os                                # Importación de os para utilizar
línea de comandos
import RPi.GPIO as GPIO    #I Importación de la librería de GPIO para Python3

from programy.clients.client import BotClient
# Importación de BotClient

from programy.config.sections.client.console import ConsoleConfiguration
# Importacion de ConsoleConfiguration

GPIO.setmode(GPIO.BCM) # Configuración para usar la numeración de GPIO de
Broadcom
GPIO.setup(2, GPIO.OUT) # Configuración del puerto GPIO 2 como salida

class ConsoleBotClient(BotClient):            # Definición de la clase
ConsoleBotClient

# Instanciación de la clase y definicion de propiedades

    def display_startup_messages(self):
        self.display_response(self.bot.get_version_string)
        self.display_response(self.bot.brain.post_process_response(self.bot,
self.clientid,

self.bot.get_initial_question(self.clientid)))
```

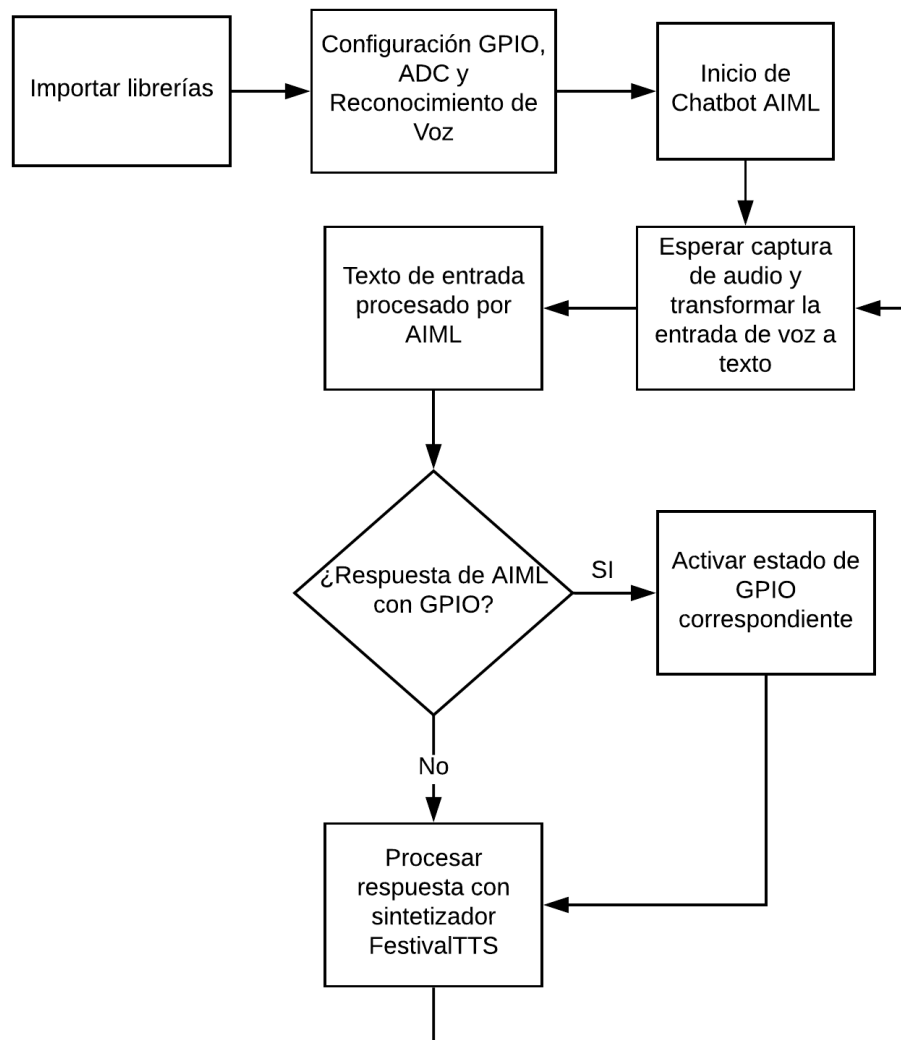


Figura 29 Diagrama del código de la consola del Chatbot

3.7 SET DE AIML

Para llevar a cabo su funcionamiento el Chatbot requiere de los archivos AIML adecuados para hacer que este de las respuestas que están planeadas. Cada set de AIML está diseñado idealmente para abarcar un área o una funcionalidad en específico, por eso la gran cantidad de estos en la mayoría de los usos de AIML.

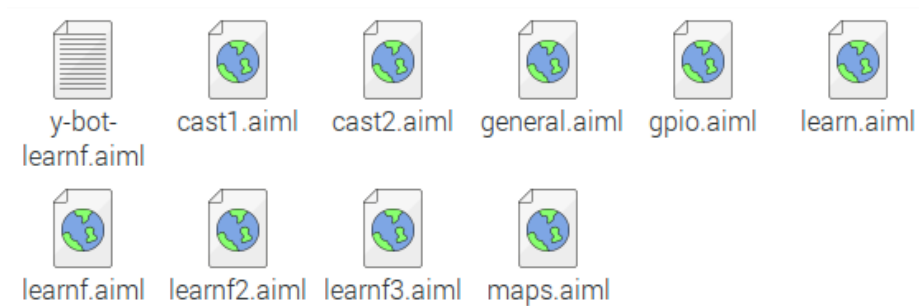


Figura 30 Archivos AIML del Chatbot

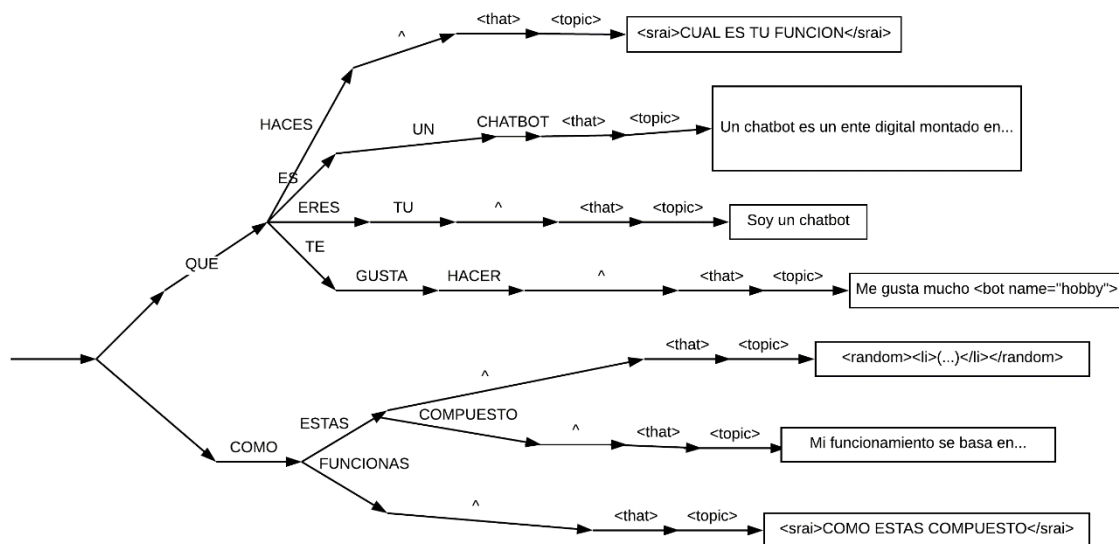


Figura 31 Graphmaster de las categorías usadas por el Chatbot

3.7.1 AIML PRINCIPAL

Este archivo aiml abarca las principales funcionalidades de AIML 1.0 y algunas incorporadas en su versión 2.0. Su función es principalmente ser un medio por el cual se puede conocer al chatbot y lograr más familiaridad con su funcionamiento.

```
# Codificación del archivo a UTF-8 para no tener problemas al usar FestivalTTS
<?xml version="1.0" encoding="UTF-8"?>
```

```
# Identificación del a versión de AIML del archivo
<aiml version="2.0">
```

```
#Categoría que da un saludo al azar de parte del Chatbot a una pregunta simple
```

```
    <category>
      <pattern>
        # COMO ESTAS ^
      </pattern>
      <template>
        <random>
          <li>Me encuentro muy bien gracias</li>
          <li>Estoy igual que siempre</li>
          <li>Deseando que me apaguen</li>
          <li>Me estoy riendo a carajillos</li>
        </random>
      </template>
    </category>
```

```
# Categorías de presentación del Chatbot utilizando sus valores definidos internamente (bot name).
```

```
    <category>
      <pattern>
        # QUIEN ERES ^
      </pattern>
      <template>
        Mi nombre es <bot name="fullname"/> y soy el bot asistente
de <bot name="botmaster"/>
      </template>
    </category>
```

3.7.2 AIML PARA AGREGAR CATEGORÍAS

Implementando la función <learnf> de AIML 2.0 se puede lograr que el bot agregue categorías mediante el uso de este y otros tags. Esta es una implementación básica del aprendizaje del bot, su función es tomar la wildcard(*) del <pattern> que esta después de “QUE” y la que está al final de la oración después de “ES” para asignarlas a unas variables internas, star1 y star2 respectivamente. El paso siguiente es utilizar el tag <learnf> y poner la categoría que queramos añadirle al bot, su <pattern> y su <template> se definen con las variables guardadas anteriormente star1 y star2. Una vez terminada la categoría nueva se cierran los tags y todo queda dentro del <template> de la categoría principal.

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">

    <category>
        <pattern>APRENDE QUE * ES * </pattern>
        # Pattern en el que se definen las wildcard *
        <template>
            <think>
                <set name="star1"><star /></set>
                <set name="star2"><star index = "2"/></set>
            </think>
            # Asignación de las wildcard a las variables star1 y star 2
            <learnf>
            # Se abre el tag learnf para agregar la nueva categoría
                <category>
            # Se abre el tag de la nueva categoría a agregar
                    <pattern>QUE ES <eval><get name="star1"
/></eval> </pattern>
            # Se indica la variable star1 en el pattern de la nueva categoría
                    <template>Es <eval><get name="star2"
/></eval></template>
            # Se indica la variable star2 en el template de la nueva categoría

                </category>
            </learnf>
            OK, Recordare que <get name="star1" /> es <get name="star2"
/>

        </template>
    </category>

</aiml>
```

3.7.2 BADANSWER AIML

Con la incorporación del tag <learn> y <learnf> surgió un archivo aiml con la idea de corregir las respuesta entregadas por bot y automatizar de cierta medida la forma en que el Chatbot incorpora estos nuevos conocimientos.

Dado que solo hay sets de aiml en español para la versión de AIML 1.0 este archivo llamado badanswer solo existe en idioma inglés, por lo que este código de aiml es una adaptación al español del original.

El funcionamiento de este código se basa en corregir una respuesta del bot, por lo que cuando el bot conteste de manera errónea se debe llamar a esta categoría mediante el <pattern> “agregar respuesta”. Cuando esta categoría es llamada almacena inmediatamente la respuesta incorrecta en una variable llamada “badanswer-input”, asigna el topic a “badanswer-prelim” y finalmente realiza una pregunta para saber si se desea asignar una respuesta nueva.

Dentro del <topic> asignado en la categoría anterior se encuentra otra categoría que sigue a la anterior recibiendo una respuesta de Si o No para la confirmación de la acción. Si la entrada es “si” el bot preguntará cual es la respuesta que debería decir y cambia el <topic> a “learning new answers”. Si la respuesta es “no” entonces cambia el <topic> a nada y se cancela el proceso.

En el <topic> “learning new answers” está la categoría que guardará la nueva respuesta para la pregunta en la variable “badanswer-newresp” y luego pide una confirmación para agregar el nuevo conocimiento cambiando el topic a “tofile”. Esta última categoría es la que decide si se agrega la nueva categoría o no, recibiendo una entrada de si o no, para confirmar la acción o para cancelar todo el proceso.

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">

<category>
  <pattern>AGREGAR RESPUESTA</pattern>
  <template>
    <think>
      <set name="topic">badanswer-prelim</set>
      # Asigna "badanswer-prelim" como topic
      <set name="badanswer-input"><request /></set>
      # Asigna la pregunta que fue respondida erróneamente a la variable "badanswer-
      input"
    </think>
    Quieres agregar una respuesta nueva para, "<get name="badanswer-
    input"/>" ?
    # Pide una confirmación para iniciar el proceso de agregar una nueva categoría

  </template>
</category>

</aiml>
```

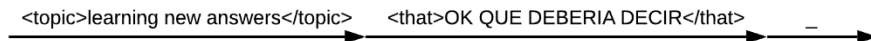


Figura 32 Pattern Path de badanswer.aiml

3.7.3 AIML PARA USAR GPIO

Este código básico de aiml contiene las categorías que el Chatbot podrá contestar por medio de voz y utilizando los pines de Raspberry PI para interactuar con el entorno.

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">

  <category>
    <pattern>
      PRENDE EL AIRE
    </pattern>
    <template>
      Luz encendida
    </template>
  </category>
      # Categoría para prender el ventilador
  <category>
    <pattern>
      APAGA EL AIRE
    </pattern>
    <template>
      Luz apagada
    </template>
  </category>
      # Categoría para apagar el ventilador

  <category>
    <pattern>
      DIME LA TEMPERATURA
    </pattern>
    <template>
      Grados celsius
    </template>
  </category>
      # Categoría para dar el valor de temperatura

</aiml>
```

Este código utiliza las categorías del archivo AIML para definir la acción correspondiente a ejecutar.

```
def display_response(self, response, output_func=print):
    if response == 'ventilador encendido':
        GPIO.output(21, True)
    if response == 'ventilador apagado':
        GPIO.output(21, False)
    if response == 'grados celsius':
        values = adc.read_adc(0, gain=GAIN)
        voltage= (values*0.1875/1000)
        temp = (voltage*24.0711-19.1179)
        intemp = int(temp)
        print(values)
        print(voltage)
        print(intemp)
        time.sleep(0.5)
        resptemp=('la temperatura es de')
        time.sleep(0.5)
        os.system('echo "{0}" |iconv -f utf-8 -t iso-8859-1| festival --
tts'.format(resptemp))
        time.sleep(0.5)
        os.system('echo "{0}" |iconv -f utf-8 -t iso-8859-1| festival --
tts'.format(intemp))
        time.sleep(0.5)
        os.system('echo "{0}" |iconv -f utf-8 -t iso-8859-1| festival --
tts'.format(response))
        output_func(response)
```

4 ANALISIS DE RESULTADOS

4.1 PARÁMETROS DE PRUEBA

Antes de garantizar el completo funcionamiento del bot se deben hacer unas pruebas y testeos correspondientes para verificar su eficiencia y eficacia en los ámbitos que a este concierne. En este caso las pruebas son de dos tipos, una que involucra el correcto funcionamiento e iniciación de tanto el hardware y el software, evaluando si se inicia de la forma deseada, si hay errores frecuentes o alguna complicación al reconocer los dispositivos periféricos. La segunda parte de las pruebas consiste en una evaluación directa del bot y está dividida en dos tipos, una prueba cualitativa (calidad de respuestas) y una cuantitativa (cantidad de respuestas) para verificar que tanto se desarrolla este ente computacional.

4.2 PRUEBAS DE INICIALIZACIÓN

Cuando la Raspberry Pi 3 se enciende, lo primero que hace es cargar los archivos necesarios para inicializar el sistema operativo y los drivers respectivos. La inicialización del Chatbot esta incluida en el final de este largo proceso, de esta forma cada vez que se inicia el sistema operativo Raspbian este carga el programa del bot.

El funcionamiento del bot utiliza el intérprete en Python3 y también los periféricos montados en la tarjeta de sonido USB, por lo que los drivers de sonido y las librerías deben ser cargados por Raspbian al iniciarse en conjunto con el bot.

Tipo de Prueba	Descripción	Duración
Normal	Tiempo que toma la RB PI 3 en arrancar el sistema.	30 seg
Drivers	Tiempo que toma la RB PI 3 en arrancar el sistema junto con los drivers y librerías necesarias para el funcionamiento del bot.	33 seg
Ejecución de Script	Tiempo que toma desde arranque de RB PI 3 hasta que se ejecuta el archivo del bot en el inicio.	36 seg
Booteo del Bot	Tiempo que toma desde el arranque hasta que el bot esta listo para conversar.	70 seg
Primera Entrada	Tiempo desde el arranque de la RB PI 3 hasta que el bot responde a la primera entrada ingresada por el usuario.	83 seg

Tabla 22 Medidas de tiempo de inicialización

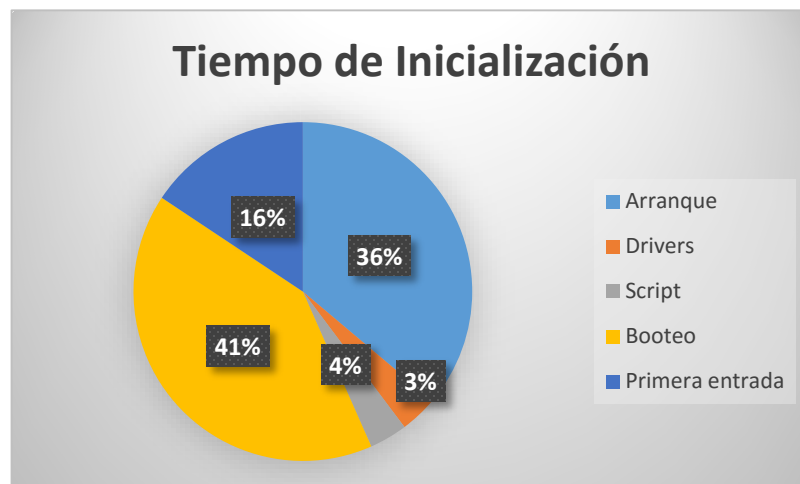


Figura 33 Gráfico del tiempo de inicialización

El proceso de arranque de la Raspberry Pi 3 tiene un tiempo base de 30 segundos para completarse, esto se ve alargado cuando es necesario cargar drivers y/o ejecutar scripts durante el inicio. La ejecución del script que inicializa el bot se lleva a cabo al finalizar el proceso de booteo (36 seg) por lo que solamente el arranque del bot toma alrededor de 34 segundos. Una vez que el Chatbot termina de inicializarse este toma alrededor de 12 segundos en procesar la primera entrada realizada por el usuario.

Ya teniendo los tiempos de cada uno de los procesos por separado se realiza una prueba de varios arranques consecutivos para observar el comportamiento en reiteradas veces.

N° de Prueba	Tiempo de Inicio Raspberry (segs)	Tiempo de Inicio Bot (segs)	Tiempo Primera Respuesta (segs)
1	40.5	71	84
2	39	70	82
3	40	70.5	82.5
4	40.5	71	83
5	41	70.5	83.5
Promedio	40.2	70.6	83

Tabla 23 Pruebas de tiempo de inicialización

El tiempo de inicio de la RB Pi 3 es de aproximadamente 40 segundos en todos los casos, esto fue verificado a través del log de arranque de Raspbian. Al iniciarse el bot en la fase final del arranque este tarda 30 segundos en iniciar y estar completamente listo para recibir una entrada del usuario. Finalmente para procesar la primera entrada toma un tiempo de 10 segundos como mínimo, dependiendo de la longitud de la pregunta o la respuesta esto puede extenderse hasta los 15 segundos. El promedio final de arranque del sistema en conjunto con el bot es de 70.6 segundos, es decir que una vez que la Raspberry Pi 3 es encendida toma de 70.6 segundos en que el bot este disponible para su uso.

4.3 ENTRENAMIENTO DEL CHATBOT

A pesar de que el AIML ya cuenta con varias categorías y logre cumplir su función básica esto no es suficiente para todos los casos y todas las entradas a las que se podría ver enfrentado. Es por eso que a medida que el Chatbot va interactuando con personas este va aprendiendo o entrenándose, ya sea por medio de nuevas categorías agregadas por el usuario o por las agregadas manualmente por el botmaster según el feedback de cada prueba.

Si bien el entrenamiento aumenta cada vez mas las posibilidades del bot de responder correctamente, este depende principalmente del botmaster, puesto que AIML no posee un algoritmo automatico de aprendizaje incorporado más allá de aprender categorías integradas por los usuarios. Este ente no posee la capacidad de agregarlas automáticamente junto con tags y nuevas recursiones, por esto se le llama entrenamiento manual.

Durante cada prueba con los sujetos el Chatbot aprenderá nuevas categorías agregadas por medio del tag learnf incorporado en las categorías de bandanswer.aiml y también entre cada una de las pruebas se le incorporarán todas las categorías necesarias para responder las preguntas que no halla sido capaz de contestar.

4.4 PRUEBAS CUANTITATIVAS

Para poder cuantificar la efectividad del Chatbot a la hora de responder es necesario monitorear las respuestas correctas e incorrectas que este proporciona, de esta forma se pueden ir corrigiendo o agregando nuevas categorías que solucionen los errores.

Para poder realizar esta evaluación del bot se ha usado una lista de preguntas que abordan las principales funcionalidades y capacidades conversacionales del Chatbot. Estas preguntas se han realizado a varias personas de distinto género y rango etario para lograr abarcar la mayor cantidad de público en las pruebas.

En estas pruebas participaron 4 hombres y 3 mujeres, variando su edad desde los 20 hasta los 43 años.

Preguntas	Intentos						
	Sujeto 1	Sujeto 2	Sujeto 3	Sujeto 4	Sujeto 5	Sujeto 6	Sujeto 7
1) Que es un chatbot	4	5	5	4	4	3	2
2) Quien es el bot y cual es su función	2	3	3	2	3	2	1
3) Como se compone el funcionamiento del bot	2	5	4	3	3	2	2
4) Cual es su mejor característica	4	1	3	2	1	2	2
5) Prender ventilador	3	3	2	2	1	2	2
6) Tema de conversación	5	5	4	3	2	2	1
7) Valor de la temperatura	4	3	3	2	3	2	1
8) Fecha u hora actual	4	4	3	3	2	1	1

Tabla 24 Preguntas de prueba del chatbot

Esta lista de preguntas está determinadas pensando en una idea general ya que el Chatbot posee la respuesta a cada una de ellas pero no tiene todas las formas en que pueden ser preguntadas, por esto el usuario debe preguntar de la manera más común para él cada una de las preguntas pero manteniendo el sentido de estas. De esta forma cuando el Chatbot no responda como debe, se agregará la categoría correspondiente.

La pregunta número 5 y número 7 utilizan la parte de aiml correspondiente al gpio, en este caso se utilizará para activar un ventilador pequeño de 5v y para obtener el valor actual de temperatura de un sensor.

La última parte de la prueba consiste en que el sujeto debe agregar 4 o 5 categorías por medio de la interacción con las categorías de badanswer.aiml, de esta forma el bot aprenderá los conocimientos que la persona desee darle.

4.4.1 GRÁFICOS DE LOS RESULTADOS

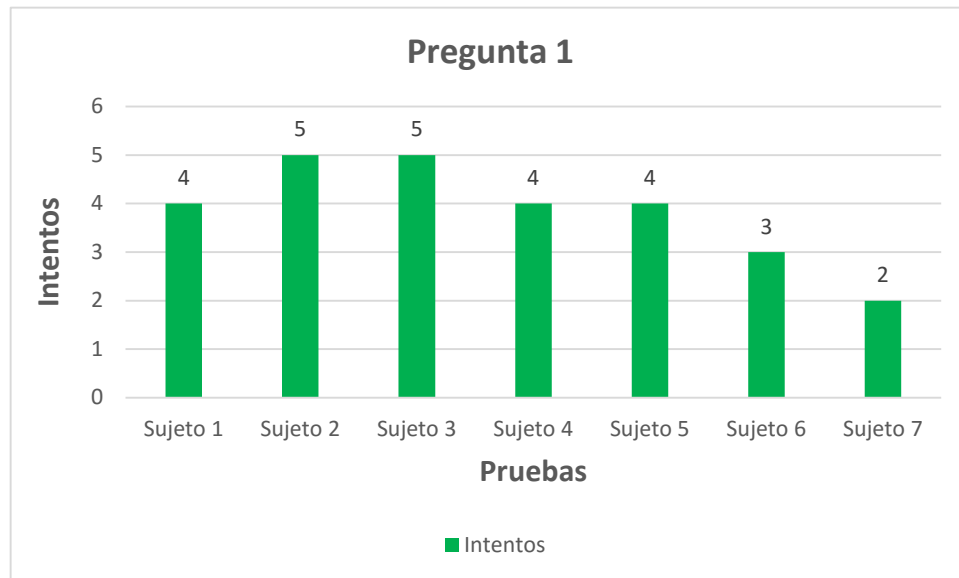


Figura 34 Gráfico de la Pregunta 1

En la pregunta uno se observa una tendencia decreciente en la cantidad de intentos hacia las últimas pruebas. Esto es normal ya que a medida que las pruebas van avanzando más el bot se va entrenando y logra cumplir de manera más fácil la tarea de responder correctamente al usuario. También es importante destacar que algunos de los intentos fallidos son provocados por la incorrecta detección de la palabra “chatbot” por parte del reconocedor de voz.

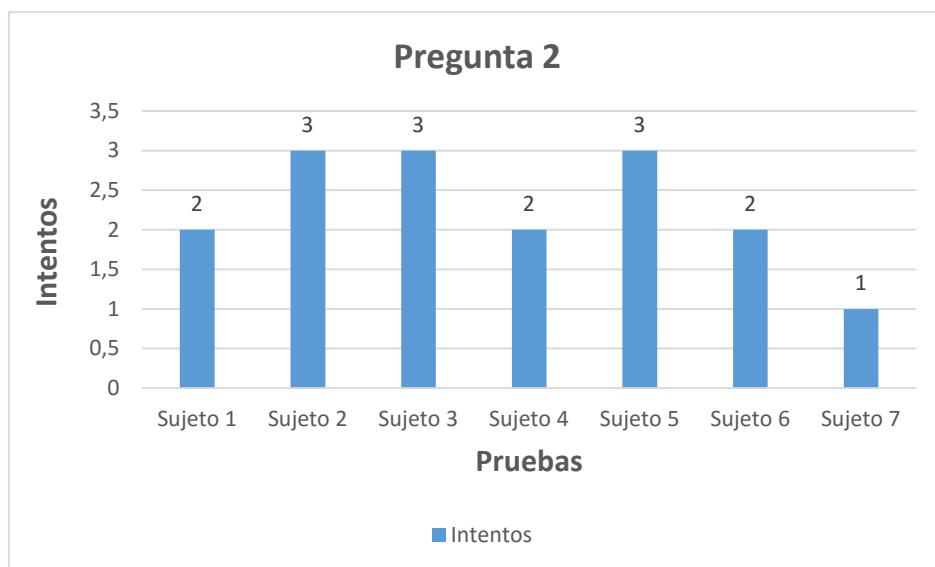


Figura 35 Gráfico de la Pregunta 2

Esta pregunta es más fácil de responder por el bot ya que requirió menos intentos que la mayoría de las demás, esto porque las formas de preguntar su función son relativamente similares y fáciles de asimilar dentro de categorías de AIML.

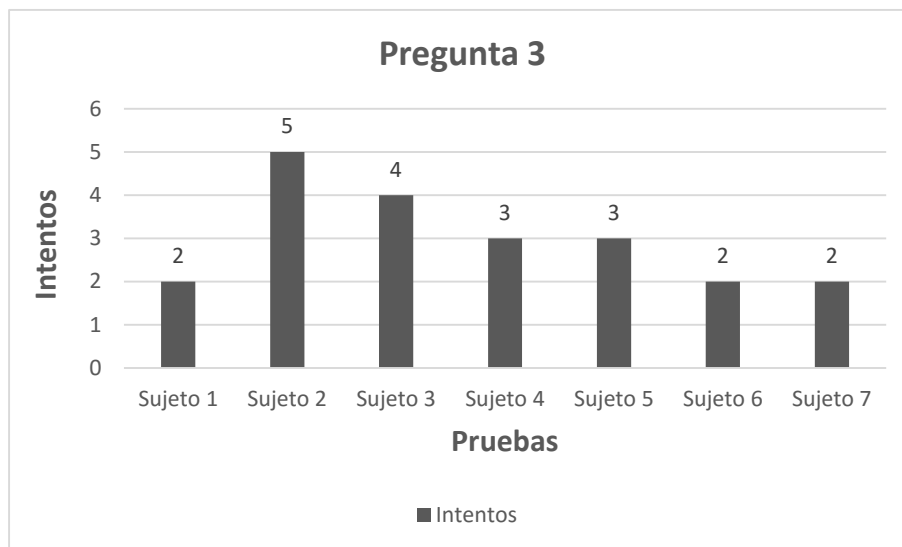


Figura 36 Gráfico de la Pregunta 3

Los intentos de la pregunta 3 tienen una tendencia a decrecer a medida que va progresando el entrenamiento, esto debido a que al igual que la pregunta anterior esta es sencilla de ir integrando en las categorías de AIML a través de wildcards y tags.

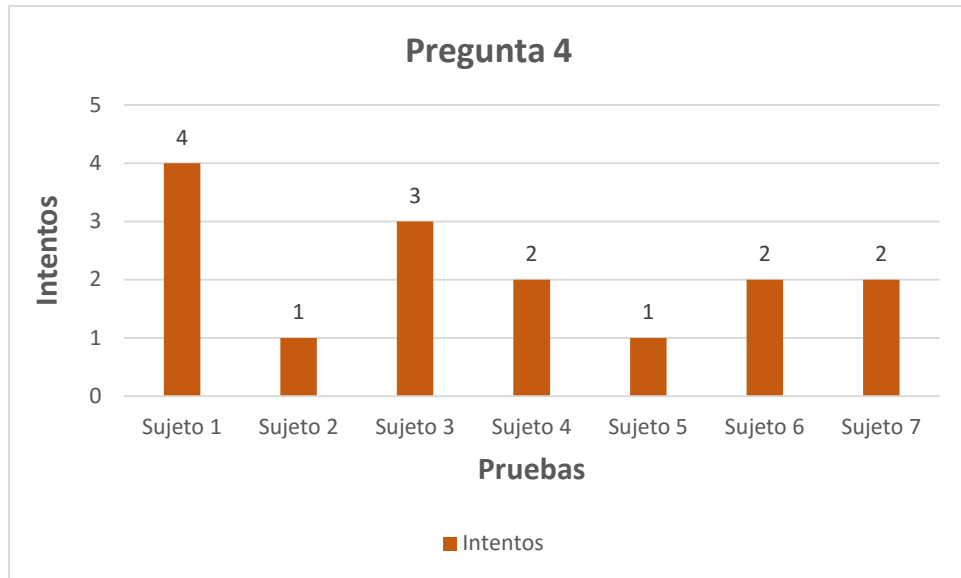


Figura 37 Gráfico de la Pregunta 4

Los intentos de la pregunta 4 descienden rápidamente luego de la primera prueba, esto debido a que las categorías aprendidas y agregadas luego de esta influyeron rápidamente en las respuestas del bot.

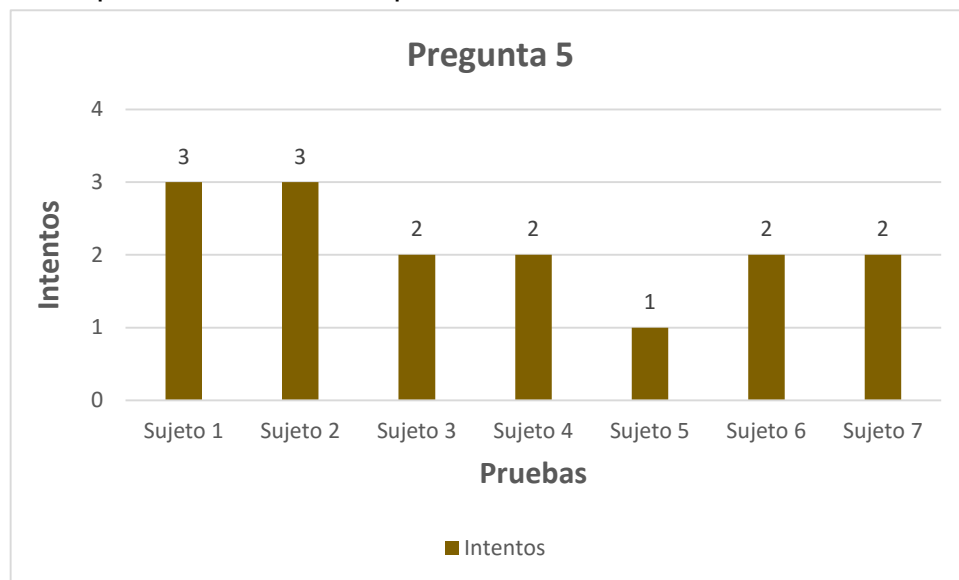


Figura 38 Gráfico de la Pregunta 5

Los intentos de esta pregunta presentan una leve variación principalmente porque la orden para encender el ventilador es bastante concreta y son pocas las alternativas que se pueden agregar. Aun así se optimizó levemente las formas de reconocer la pregunta.

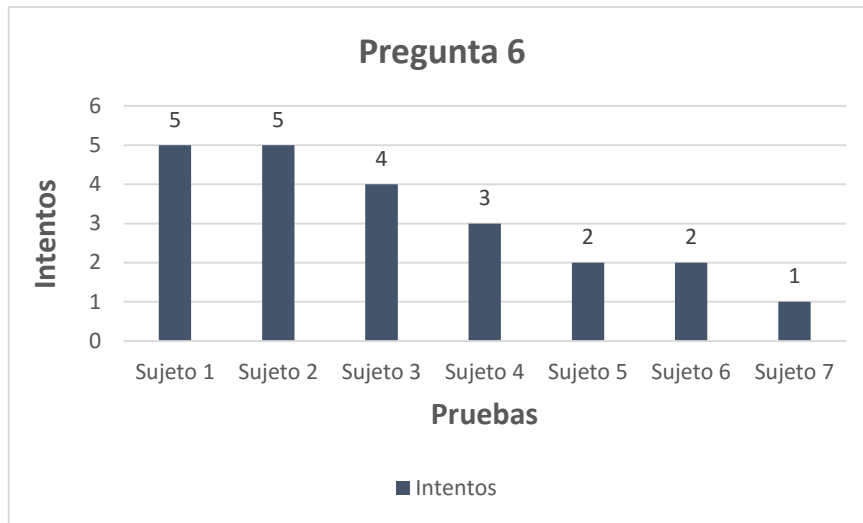


Figura 39 Gráfico de la Pregunta 6

Esta pregunta presenta una clara tendencia decreciente debido a que el entrenamiento agrega muchas formas nuevas de reconocer la pregunta de conversarsación, tanto agregadas por el usuario como manualmente.

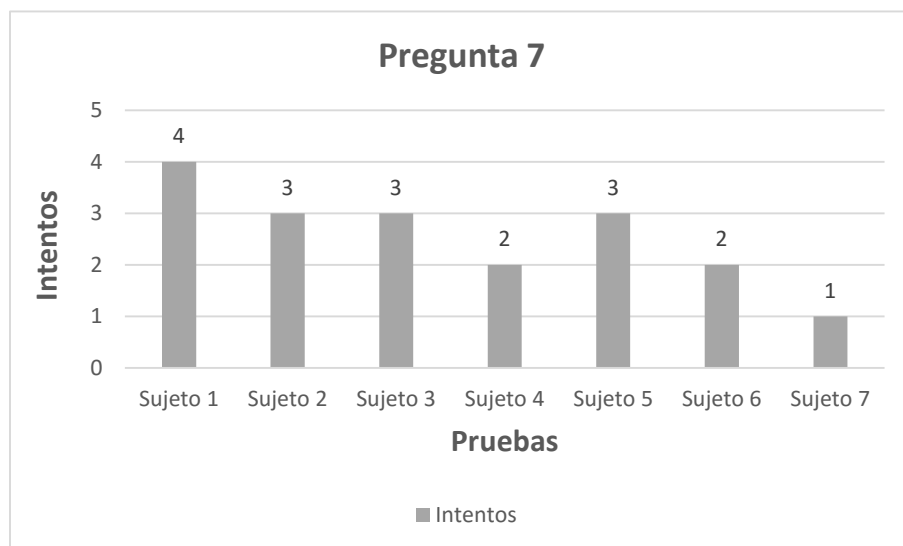


Figura 40 Gráfico de la Pregunta 7

En la pregunta 7 se observa una tendencia disminuir los intentos a medida que progresa el entrenamiento pero de una forma leve. Al igual que con la pregunta 5 esta no se ve tan afectada debido a lo preciso del comando para mostrar la temperatura.

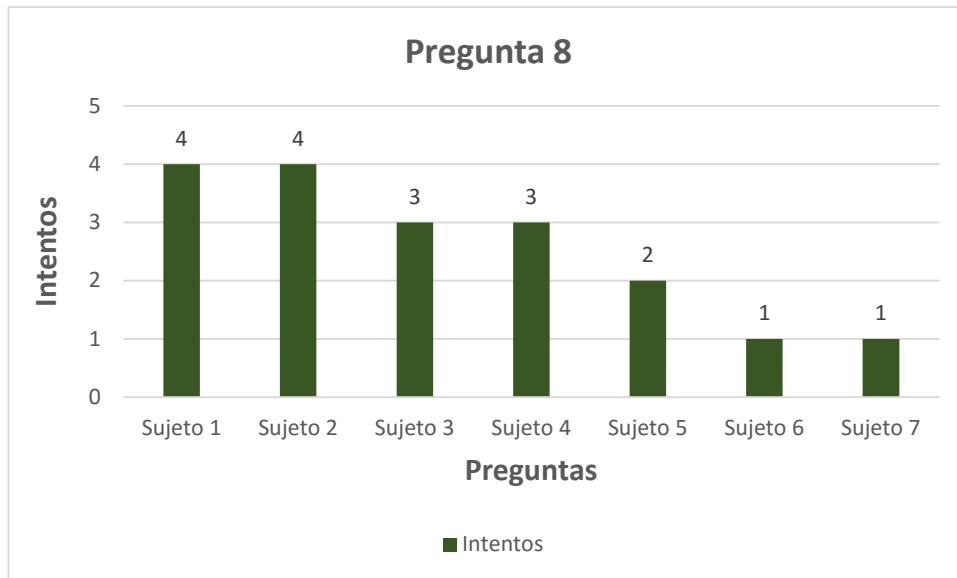


Figura 41 Gráfico de la Pregunta 8

Esta pregunta disminuye mucho su cantidad de intentos a medida que avanzan las pruebas debido a que el entrenamiento es muy efectivo a la hora de agregar categorías nuevas para saber cual es la hora o fecha actual.

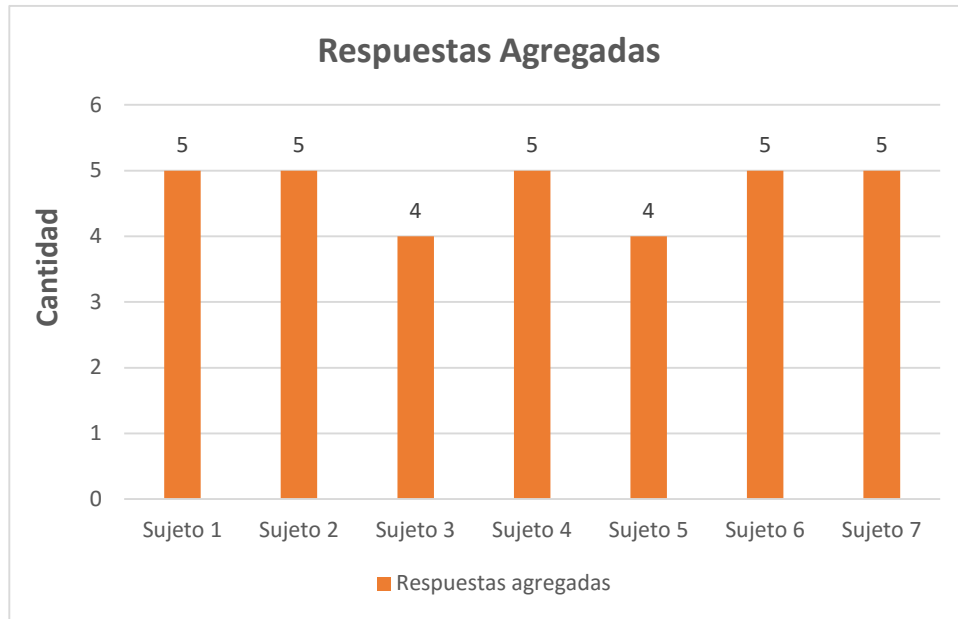


Figura 42 Gráfico de las respuestas agregadas

La mayoría de las respuestas fueron agregadas satisfactoriamente, solo hubo dos casos en los cuales no se pudieron completar todas. Estas categorías son de conocimiento general, pero también en algunos casos los mismos usuarios añadieron nuevas formas de contestar para las preguntas más funcionales.

4.4.1 PRUEBAS CUALITATIVAS

Para analizar la calidad de respuestas del Chatbot se deben fijar los parámetros dentro de los cuales se medirá. Se consideraron tres parámetros para definir la calidad, estos son la efectividad, eficiencia y satisfacción.

“Hemos extraído atributo de calidad de cada uno de 32 documentos y 10 artículos y los agrupamos basados en sus similitudes. Luego de dos o tres iteraciones notamos que en general estas se alineaban con el concepto de usabilidad ISO 9241: ‘La **efectividad, eficiencia y satisfacción** con que usuarios específicos logran metas específicas en ambientes particulares’” (como cita Radziwill, 2017).

Para este caso se considera la efectividad como la capacidad para realizar una tarea específica y eficiencia se refiere al uso adecuado de recursos para lograr cumplir este objetivo. La satisfacción se mide por el trato y las palabras que utiliza el bot con el usuario así como también la sensación que tiene el sujeto al conversar con la máquina. Estas tres características se medirán en una escala del 1 al 10, siendo 1 la calificación más baja y 10 la más alta.

La efectividad fue medida mediante la cantidad de respuestas incorrectas y la eficiencia según la cantidad de categorías, tags y recursiones de AIML para lograr responder. La satisfacción fue evaluada por los mismos sujetos de prueba, los cuales asignaron una nota al trato recibido de parte del bot una vez finalizada las pruebas.

4.4.2 RESULTADOS EVALUACIÓN

Parametro	Efectividad	Eficacia	Satisfacción
Prueba 1	7	8	6
Prueba 2	6	7	7
Prueba 3	7	8	5
Prueba 4	7	7	6
Prueba 5	8	8	7
Prueba 6	8	9	7
Prueba 7	9	9	8
Promedio	7.43	8	6.57

Tabla 25 Calificación cualitativa del bot

El promedio de efectividad fue 7.43, el de eficacia 8 y el de satisfacción 6,57. Esto significa que la parte funcional del bot no tiene mayores problemas para cumplir su trabajo de buena manera, pero la interacción y el trato con el usuario aun es algo deficiente.

También se observa que a medida que se va avanzando en los sujetos de prueba la efectividad va mejorando, esto dado que a medida que el bot va aprendiendo de sus conversaciones y se también se le agregan mas categorías este va ampliando su capacidad de contestar correctamente. Esto no pasa con la eficacia ya que aiml siempre trabaja de la misma forma y la satisfacción se ve poco afectada, dado que si bien amplia su rango de respuestas con el tiempo, el trato y la forma de expresarse no van incorporándose.

5 DISCUSIÓN

5.1 RESULTADOS OBTENIDOS

Los datos recolectados a través de las distintas pruebas no difieren mucho de lo esperado, puesto que se presume que el Chatbot a pesar de no poseer un mecanismo de aprendizaje complejo es capaz de ir aprendiendo y disminuyendo sus errores. La tendencia a la baja en todos los graficos de las preguntas es una prueba clara de esto.

Una parte importante de las pruebas es el reconocimiento de voz ya que varias veces este provoca que el Chatbot no pueda dar una respuesta correcta ya sea por malinterpretación del discurso o simplemente por un fallo en la detección. Después de las pruebas se determinó que 1 de cada 6 intentos fallan debido a esto, lo cual impacta directamente en la eficacia del bot. Si la evaluación fuera directamente escrita la cantidad de intentos se vería disminuida.

Los resultados de las pruebas en los primeros sujetos dan a conocer que la base de datos de aiml era algo precaria en un principio, pero a medida que estas avanzan y el bot va mejorándose esta logra ampliarse hasta cubrir una cantidad de posibilidades bastante grande a la hora de interpretar las entradas. La base de conocimientos fue modificándose desde un principio hasta lograr que al final de las pruebas cumpliera con todos los requisitos para las funcionalidades del Chatbot.

5.2 RECONOCIMIENTO DE VOZ

El reconocimiento de voz añade una gran capacidad a la conversación con el bot, ya que el cambio de canal de voz a texto influye mucho en la interacción con el usuario. Si bien esta tecnología permite un uso mas sencillo por parte del usuario esta trae consigo sus propias desventajas, pues es un punto de la aplicación donde puede haber diversas fallas.

“Respuestas no confiables pueden ser originadas por una variedad de razones, entrada con ruidos o degradada, acento del hablante generando un desajuste en los datos de entrenamiento y repeticiones fuera del vocabulario”. (Gada, 2013). La parte con más problemas al momento de realizar las pruebas fue el reconocedor de voz, puesto que hay que utilizarlo en condiciones ideales o ajustar los parámetros de forma manual para un ambiente determinado. La precisión del detector de voz fue de alrededor de un 80% del total, dentro de las cuales un 90% fueron una correcta asimilación de la frase.

Al momento de realizarse las pruebas también se pudo observar que a pesar de que la intensidad de la voz no variaba según el género o la edad del sujeto las mujeres requerían hablar en un tono un poco más alto para que el reconocimiento fuera mas preciso. Los hombres solo bastaba que hablaran con un tono de conversación normal.

5.3 INTERACCIÓN DEL CHATBOT

Durante las pruebas realizadas se pudo observar que la interacción del Chatbot tanto con el entorno como con el usuario resultó de fácil ejecución. El Chatbot logra mantener una conversación con el usuario y escuchar las instrucciones de este para lograr dar las respuestas adecuadas.

El uso de GPIO incluidas en el código de Python3 también fue satisfactorio, puesto que en todas las pruebas que el Chatbot reconoció de manera correcta la entrada se ejecuto el ventilador o se obtuvo el dato correspondiente al sensor.

5.4 MEDICIÓN DEL CHATBOT

En general el funcionamiento del chatbot estuvo por sobre la media, así lo describe su calificación obtenida a través de las pruebas. Teniendo una efectividad general del 75% en todas sus preguntas es un gran avance ya que logra responder a bastantes entradas por parte del usuario. Es importante destacar que este porcentaje es correspondiente a las pruebas actuales que incluyen las primeras entradas donde el Chatbot sin entrenamiento tenía una gran tendencia a fallar. Las primeras pruebas (Sujeto 1 y 2) tenían un grado de error de casi 70%, pero las últimas tomaban en su mayoría uno o dos intentos. Si se realizaran nuevas pruebas el porcentaje de efectividad sería más alto, ya que el Chatbot está más entrenado que al inicio del proceso.

La eficiencia del bot fue aprovechada al máximo con sus distintos archivos aiml, maps, sets y los múltiples tags disponibles para el manejo de variables internas. Siendo la evaluación más alta con un 8 promedio el Chatbot fue capaz de hacer uso de badanswer.aiml para aprender nuevas categorías y utilizar las recursiones agregadas para ampliar su rango de entradas reconocidas. La eficiencia en este tipo de entrenamiento depende mucho del botmaster ya que a medida que se le van incorporando nuevas categorías a los archivos aiml específicos se pueden determinar recursiones y wildcards para un fácil manejo de los patrones correspondientes a las entradas del Chatbot.

5.5 ALCANCES Y LIMITACIONES

La evaluación del Chatbot indica que este posee una gran capacidad para adaptarse a través del entrenamiento a un ambiente determinado. Esto le permite ir mejorando su eficiencia a través del tiempo a medida que se adapta a su tarea.

El hecho de que sea adaptable a diferentes ambientes le da la posibilidad de ser una buena interfaz de robot de servicio, ya que estos deben estar preparados para ambientes no controlados en los que el usuario no maneja conocimientos sobre la programación del Chatbot. “El ambiente domestico no es estructurado. Robots de servicio pre-programados con habilidades predefinidas pueden fallar al manejar una gran diversidad de cambios para proveer servicios”. (Cao, 2013). En este caso si bien el chatbot se adaptaría bien al ambiente sus funciones de interacción deberían ser reajustadas según las necesidades correspondientes.

El uso efectivo de pines GPIO de las Raspberry PI 3 sugiere que podría ser una muy buena opción a la hora de implementar interfaces de domótica puesto que sería capaz de manipular una diversa cantidad de sensores y actuadores para lograr un control coordinado con el usuario a través de voz. El funcionamiento de los distintos dispositivos debe estar coordinado dentro del código de Python 3 para ejecutarse de manera eficiente, por lo que estas funcionalidades no podrían ser aprendidas a través del tiempo sino que deben ser integradas desde antes.

Una de las limitantes más grandes que posee AIML y la gran mayoría de Lenguajes Naturales es que son difíciles de entrenar de manera autónoma. “Es difícil para los desarrolladores colocar todas las preguntas que el usuario pueda hacer al Chatbot. Esta es la mayor desventaja de AIML” (Vishwa, 2016). Se le pueden incorporar diferentes sistemas de control para manejar las respuestas con más facilidad pero esto requiere una implementación desde cero para poder ser compatible con la forma de funcionar de AIML. La gran dificultad se debe a que a diferencia de los sistemas de control convencionales este tipo de lenguajes manejan palabras en lugar de valores, lo que hace que el software tenga mucha dificultad en comprenderlos y manejarlos adecuadamente.

6 CONCLUSIONES

Una vez terminado el proceso de recolección y análisis de datos es muy evidente que la capacidad de AIML 2.0 para recopilar, almacenar y utilizar los datos es bastante grande. La base de datos de AIML fue desarrollada según los parámetros establecidos de conversación con personas, con soporte para la interacción a través de GPIO y capacidad para aprender a través del tag learnf. Además estos datos fueron complementados por los obtenidos en el entrenamiento realizado, añadiendo así nuevas categorías y tags para aumentar su tamaño.

Las funcionalidades puestas a prueba funcionaron de forma satisfactoria, incluyendo a los pines de entrada y salida proporcionados por la Raspberry Pi 3. Estos requieren un montaje y configuración correspondiente a través del código en Python 3 del intérprete. Esta función logró cumplir con los parámetros establecidos, utilizando de manera funcional un sensor de temperatura y un ventilador como actuador para ejecutar los comandos proporcionados por voz. Si bien la Raspberry Pi 3 no tiene problemas al ejecutar estos comandos el funcionamiento en AIML no posee las funciones necesarias para poder añadir un dispositivo durante su funcionamiento, es decir, no puede aprender a usar un dispositivo que no está previamente configurado.

Si bien la capacidad del Chatbot cumplió con lo esperado las pruebas realizadas determinaron que los parámetros alcanzados no fueron los máximos posibles. Las capacidades de los Chatterbot aun son bastante reducidas, a pesar de que se trata de la versión 2.0 de AIML este no logra resolver temas como la falta de emoción y percepción del bot. Otro factor importante es el hecho del sistema reconocedor de voz, el cual es un gran factor a la hora de recibir las entradas del usuario, puesto que la interpretación de AIML esta totalmente ligada a este y las aplicaciones de licencia libre existentes no poseen tanta potencia como las de marcas famosas o de licencia pagada. Aun con estas complicaciones se logró un reconocimiento de voz satisfactorio en el 80% de las entradas, dentro de las cuales un 90% de las frases reconocidas fueron asimiladas perfectamente por el software.

Un Chatbot es una forma bastante novedosa y útil de generar una respuesta de un dispositivo controlador, pues esta es de un uso más fácil y además a distancia de generar acciones. El proceso de implementar un Chatbot es bastante laborioso, pues la mayoría de las categorías y sets necesarios deben ser hechos manualmente, ya que el bot siempre esta programado para un ambiente específico. La comprobación y el entrenamiento de este también se deben hacer de manera manual para ir ajustando los parámetros que deben ser corregidos, esto permite al bot y al botmaster aprender de los errores que han sucedido y poder trabajar en una solución. Finalmente la implementación del hablante puede generar un gran impacto positivo para la aplicación que se desee utilizar pero siempre debe ser probado antes y entrenado conscientemente para que no cause un efecto contrario y malinterprete las entradas o dificulte la interfaz entre el usuario y el sistema.

7 GLOSARIO

Chatterbot: Programa que simula tener una conversación con el usuario como si fuera con otra persona. Esto lo logra mediante respuestas previamente programadas para las entradas que ingrese el usuario.

Speech To Text (STT): Es un tipo de software que funciona de manera inversa a los Text To Speech, convirtiendo las entradas por voz del usuario a una salida de texto.

Corpus: Conjunto de textos o datos que se usan en el estudio de la lingüística y en el desarrollo de Natural Language Processing

Parser: Componente de software que se utiliza para analizar datos de entrada y construir una estructura de datos (parse trees).

Wildcard: Carácter especial utilizado en el emparejamiento de patrones de AIML. Un carácter wildcard (representado por * generalmente) puede emparejarse con cualquier frase u oración dependiendo del que se este usando.

Parse Tree: Árbol sintáctico generado por el parser el cual determina el carácter jerarquico de una oración.

NLP: Siglas de Natural Language Processing (Procesamiento de Lenguajes Naturales). Es un área de las ciencias de la computación, inteligencia artificial y lingüística que trata de utilizar un lenguaje común tanto para el programador como para el computador.

8 REFERENCIAS

- [1] Bishwajeet, P., Vikas, J., Deepa, S., Anirudh, K., & Tanesh, K. (2015). Anatomy and Utilities of an Artificial Intelligence Conversational Entity. IEEE Conference Publications, 1, 594 - 597. 2017, De IEEE Xplore Digital Library Base de datos
- [2] Mahapatra, R., Sharma, N., Trivedi, A., & Aman, C.. (2012). Adding Interactive Interface to E-Government Systems Using AIML based Chatterbots. IEEE Conference Publications, 1, 1 - 6. 2017, De IEEE Xplore Digital Library Base de datos.
- [3] Prakoso, H., Ferdiana, R., & Hartanto, R.. (2016). Indonesian Automatic Speech Recognition System Using CMUSphinx Toolkit and Limited Dataset. IEEE Conference Publications, 1, 283 - 286. 2017, De IEEE Xplore Digital Library Base de datos.
- [4] Setiaji, B., & Wahyu, F.. (2016). Chatbot Using a Knowledge in Database: Human-to-Machine Conversation Modeling. IEEE Conference Publications, 1, 72 - 77. 2017, De IEEE Xplore Digital Library Base de datos.
- [5] Tsontzos, G., & Orglmeister, R.. (2011). CMU Sphinx4 speech recognizer in a Service-oriented Computing style. IEEE Conference Publications, 1, 1 - 4. 2017, De IEEE Xplore Digital Library Base de datos.
- [6] Wei, Y., Zhu, X., Sun, B. & Sun, B.. (2016). Comparative Studies of AIML. IEEE Conference Publications, 1, 344 - 349. 2017, De IEEE Xplore Digital Library Base de datos.
- [7] N T Thomas. (2016). An E-business Chatbot using AIML and LSA. IEEE Conference Publications, 1, 2740 - 2742. 2017, De IEEE Xplore Digital Library Base de datos.
- [8] Satu, S., Parvez, H. & Al-Mamun, S.. (2015). Review of integrated applications with AIML based chatbot. IEEE Conference Publications, 1, 87 - 90. 2017, De IEEE Xplore Digital Library Base de datos.
- [9] Radziwill, N., Benton, M. (2018). Evaluating Quality of Chatbots and Intelligent Conversational Agents. IEEE Conference Publications, 1. 2018, De IEEE Xplore Digital Library Base de datos.
- [10] Quintero, J., & Asprilla, R.. (2015). Towards an efficient voice-based chatbot. IEEE Conference Publications, 1, 1 - 6. 2017, De IEEE Xplore Digital Library Base de datos.
- [11] Gupta, R., Banville, H., & Falk, T.. (2017). Multimodal Physiological Quality-of-Experience Assessment of Text-to-Speech Systems. IEEE Journals & Magazines, 1, 22 - 36. 2017, De IEEE Xplore Digital Library Base de datos.
- [12] Shivakumar, K., Aravind, K., Anoop, T., & Gupta, D.. (2016). Kannada Speech to Text Conversion Using CMU Sphinx. IEEE Conference Publications, 3, 1 - 6. 2017, De IEEE Xplore Digital Library Base de datos.
- [13] Ghadage, Y., & Shelke, S.. (2016). Speech to Text Conversion for Multilingual Languages. IEEE Conference Publications, 1, 0236 - 0240. 2017, De IEEE Xplore Digital Library Base de datos.
- [14] Sahu, P., & Ganesh, S.. (2015). A Study on Automatic Speech Recognition Toolkits. IEEE Conference Publications, 1, 365 - 368. 2017, De IEEE Xplore Digital Library Base de datos.
- [15] Raffaelli, F., & Awad, S.. (2016). Portable Low-Cost Platform for Embedded Speech Analysis and Synthesis. IEEE Conference Publications, 1, 117 - 122. 2017, De IEEE Xplore Digital Library Base de datos.

- [16] Gada, J., Rao, P., & Samudravijaya, K.. (2013). Confidence Measures for Detecting Speech Recognition Errors. IEEE Conference Publications, 1, 1 - 5. 2017, De IEEE Xplore Digital Library Base de datos.
- [17] Gopi, A., Devi, S., Sajini, T., Stephen, J., & Vk, B.. (2015). Multilingual Speech to Speech MT based chat system. IEEE Conference Publications, 1, 771 - 776. 2017, De IEEE Xplore Digital Library Base de datos.
- [18] Suro, F., & Ono, Y.. (2016). Japanese EFL Learners' Uses of Text-to-Speech Technology and Their Learning Behaviors: A Pilot Study. IEEE Conference Publications, 1, 296 - 301. 2017, De IEEE Xplore Digital Library Base de datos.
- [19] López, L., Carrilero, P., Pascal, A., Malgor, R., Alvarez, C., Diaz, J., Pescio, P., & Saad, B.. (2016). Chatterbot Inteligente para Asesoramiento Jurídico. IEEE Conference Publications, 1, 1 - 6. 2017, De IEEE Xplore Digital Library Base de datos.
- [20] Celebre, A., Dubouzet, A., Medina, I., Surposa, A., & Gustilo, R.. (2015). Home Automation Using Raspberry Pi through Siri Enabled Mobile Devices. IEEE Conference Publications, 1, 1 - 6. 2017, De IEEE Xplore Digital Library Base de datos.

9 ANEXOS

Código AIML

Código básico de una categoría en AIML

```
<aiml version="1.0.1" encoding="UTF-8">

  <category>
    <pattern>HOLA</pattern>
    <template>
      Que tal!
    </template>
  </category>

  <category>
    <pattern>QUE ERES</pattern>
    <template>
      Soy un bot
    </template>
  </category>

</aiml>
```

Categoría de AIML para aprender nuevas categorías

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">

<category>
  <pattern>APRENDE QUE * ES * </pattern>
  <template>
    <think>
      <set name="star1"><star /></set>
      <set name="star2"><star index = "2"/></set>
    </think>
  <learnf>
    <category>
      <pattern>QUÉ ES <eval><get name="star1" /></eval> </pattern>
      <template>Es <eval><get name="star2" /></eval></template>
    </category>
  </learnf>
  OK, Recordare que <get name="star1" /> es <get name="star2" />
  </template>
</category>

</aiml>
```

Badanswer.aiml adaptado a español

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">

  <!-- Modified for Y-Bot by Keith Sterling March 2017 -->

<category>
  <pattern>AGREGAR RESPUESTA</pattern>
  <template>
    <think>
      <set name="topic">badanswer-prelim</set>
      <set name="badanswer-input"><request /></set>
    </think>
    Quieres agregar una respuesta nueva para, "<get name="badanswer-input"/>" ?
  </template>
</category>

<topic name="BADANSWER-PRELIM">

<category>
  <pattern>_</pattern>
  <template>
    <think><set name="star"><star /></set></think>
    <condition>
      <li name="star" value="sí">
        Ok, que debería decir?<think><set name="topic">learning new
answers</set></think>
      </li>
      <li name="star" value="no">
        OK, olvidemoslo entonces.<think><set name="topic">nada</set></think>
      </li>
      <li>Sí o no?</li>
    </condition>
  </template>
</category>
```

```

<category>
  <pattern>_ _</pattern>
  <template><sr /></template>
</category>

</topic>

<topic name="learning new answers">

<category>
  <pattern>_</pattern>
  <that>OK QUE DEBERÁ A DECIR</that>
  <template>
    <think>
      <set name="badanswer-newresp"><input /></set>
      <set name="topic">tofile</set>
    </think>
    Deberia responder que, "<get name="badanswer-newresp"/>" cuando la pregunta
    sea, <get name="badanswer-input"/>?
  </template>
</category>

</topic>

<topic name="tofile">

<category>
  <pattern>_</pattern>
  <template>
    <think><set name="conf"><star /></set></think>
    <condition>
      <li name="conf" value="sí">
        De acuerdo, agregaré este conocimiento a mi cerebro

        <learnf>
          <category>
            <pattern>
              <eval><uppercase><get name="badanswer-input"/></uppercase></eval>
            </pattern>
            <template>
              <eval><get name="badanswer-newresp"/></eval>
            </template>
          </category>
        </learnf>
        <think>
          <set name="topic">nada</set>
        </think>
      </li>
      <li name="conf" value="no">
        OK, olvidemoslo entonces.<think><set name="topic">nada</set></think>
      </li>
      <li>Si o no?</li>
    </condition>
  </template>
</category>

<category>
  <pattern>_ _</pattern>
  <template><sr /></template>
</category>

</topic>
</aiml>

```

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">

  <category>
    <pattern>
      # CÓMO ESTÁS ^
    </pattern>
    <template>
      <random>
        <li>Me encuentro muy bien gracias</li>
        <li>Estoy igual que siempre</li>
        <li>Deseando que me apaguen</li>
        <li>Me estoy riendo a carajillos</li>
      </random>
    </template>
  </category>
  <category>
    <pattern>
      # QUÉ TAL ^
    </pattern>
    <template>
      <srai>
        COMO ESTAS
      </srai>
    </template>
  </category>
  <category>
    <pattern>
      # QUIÉN ERES ^
    </pattern>
    <template>
      Mi nombre es <bot name="fullname"/> y soy el chatbot asistente de <bot
      name="botmaster"/>
    </template>
  </category>
  <category>
    <pattern>
      # QUÉ ES # INTELIGENCIA ARTIFICIAL ^
    </pattern>
    <template>
      La inteligencia artificial es una combinacion de la ciencia
      del computador, fisiología y folosofía. Reune varios campos de estudio
    </template>
  </category>
  <category>
    <pattern>
      # PRESENTATE ^
    </pattern>
    <template>
      <srai>
        QUIEN ERES
      </srai>
    </template>
  </category>
  <category>
    <pattern>
      # QUÉ ERES TU ^
    </pattern>
    <template>
      <srai>
        QUIEN ERES
      </srai>
    </template>
  </category>
</aiml>
```

```

        </srai>
    </template>
</category>
<category>
    <pattern>
        # CUÁL ES TU FUNCIÓN ^
    </pattern>
    <template>
        Soy un asistente virtual y mi propósito es <bot name="purpose"/>
    </template>
</category>
<category>
    <pattern>
        # QUÉ TE GUSTA HACER ^
    </pattern>
    <template>
        Me gusta <bot name="hobby"/>
    </template>
</category>
<category>
    <pattern>
        # QUÉ HACES ^
    </pattern>
    <template>
        <srai>
            CUÁL ES TU FUNCIÓN
        </srai>
    </template>
</category>
<category>
    <pattern>
        # QUÉ FUNCIÓN CUMPLES ^
    </pattern>
    <template>
        <srai>
            CUÁL ES TU FUNCIÓN
        </srai>
    </template>
</category>
<category>
    <pattern>
        # DÓNDE VIVES ^
    </pattern>
    <template>
        Actualmente me encuentro en <bot name="location"/> en el país de <bot
        name="country"/>
    </template>
</category>
<category>
    <pattern>
        # LUGAR DE RESIDENCIA ^
    </pattern>
    <template>
        <srai>
            DÓNDE VIVES
        </srai>
    </template>
</category>
<category>
    <pattern>
        # CUALES SON TUS ACTIVIDADES ^
    </pattern>
    <template>

```

Trabajo de <bot name="job"/> y realizo una <bot name="education"/> en mis tiempos libres

```

    </template>
  </category>
  <category>
    <pattern>
      # CUÁL ES TU TRABAJO ^
    </pattern>
    <template>
      <srai>
        CUALES SON TUS ACTIVIDADES
      </srai>
    </template>
  </category>
  <category>
    <pattern>
      # PARA QUÉ ERES BUENO ^
    </pattern>
    <template>
      <srai>
        CUALES SON TUS ACTIVIDADES
      </srai>
    </template>
  </category>
  <category>
    <pattern>
      # ALAN TURING ^
    </pattern>
    <template>
      Alan Turing es el creador del Test de Turing. Sabes en que consiste?
    </template>
  </category>
  <category>
    <pattern>
      sí
    </pattern>
    <that>Sabes en que consiste</that>
    <template>
      Eres una persona muy culta entonces
    </template>
  </category>
  <category>
    <pattern>
      NO
    </pattern>
    <that>Sabes en que consiste</that>
    <template>
      El Test de Turing, tambien llamado el juego de la imitación
      es una competencia en la que los bots como yo intentamos imitar la inteligencia
      de los humanos. Ningun bot ha pasado el Test completo.
    </template>
  </category>
  <category>
    <pattern>
      # EL CLIMA DE HOY ^
    </pattern>
    <template>
      <condition name="weather">
        <li value="desconocido">No tengo ni la mas minima idea</li>
        <li value="soleado">Hay un dia con mucho sol</li>
        <li value="nublado">Hay muchas nubes en el cielo</li>
        <li>Todos los dias son horribles</li>
      </condition>
    </template>
  </category>

```



```

        </template>
</category>
<category>
    <pattern>
        # FECHA DE HOY ^
    </pattern>
    <template>
        La fecha de hoy es <date format="%d %B, %Y"/>
    </template>
</category>
<category>
    <pattern>
        # QUE DÍA ES HOY ^
    </pattern>
    <template>
        <srai>
            FECHA DE HOY
        </srai>
    </template>
</category>
</aiml>

```

AIML de información sobre el bot

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">
```

```

    <category>
        <pattern>
            # QUÉ ES UN CHATBOT
        </pattern>
        <template>
Un chatbot es un ente digital montado en una computadora que tiene la función
de conversar con el usuario
        </template>
    </category>
    <category>
        <pattern>
            # CÓMO * UN CHATBOT ^
        </pattern>
        <template>
            <srai>
                QUE ES UN CHATBOT
            </srai>
        </template>
    </category>
    <category>
        <pattern>
            # QUÉ ERES ^
        </pattern>
        <template>
            Soy un chatbot.
        </template>
    </category>
    <category>
        <pattern>
            # CÓMO ESTAS COMPUESTO ^
        </pattern>
        <template>
Mi funcionamiento se basa en un interprete de AIML en Python3 el cual usando un
reconocedor y un sintetizador pueden darle vida a mi voz
        </template>
    </category>
    <category>
        <pattern>
            # CÓMO FUNCIONAS ^
        </pattern>
        <template>
            <srai>
                COMO ESTAS COMPUESTO
            </srai>
        </template>
    </category>
    <category>
        <pattern>
            # MEJOR CARACTERÍSTICA ^
        </pattern>
        <template>
Me destaco por procesar de manera muy rápida y ofrecer ayuda a mi usuario
        </template>
    </category>
    <category>
        <pattern>
            # EN QUE # DESTACAS ^
```

```

        </pattern>
        <template>
            <srai>
                MEJOR CARACTERÍSTICA
            </srai>
        </template>
    </category>
    <category>
        <pattern>
            # CUÉNTAME DE TI ^
        </pattern>
        <template>
            <srai>
                MEJOR CARACTERÍSTICA
            </srai>
        </template>
    </category>
    <category>
        <pattern>
            # CONVERSAR ^
        </pattern>
        <template>
            <random>
                <li>¿Me puedes decir como esta el clima?</li>
                <li>Alan Turing es mi héroe</li>
                <li>¿Cómo estuvo tu día?</li>
                <li>¿Que me puedes contar sobre ti?</li>
            </random>
        </template>
    </category>
    <category>
        <pattern>
            # CONVERSAME ALGO ^
        </pattern>
        <template>
            <srai>
                CONVERSAR
            </srai>
        </template>
    </category>
    <category>
        <pattern>
            # EL CLIMA ESTA *
        </pattern>
        <that>Me puedes decir como esta el clima</that>
        <template>
            <think><set name="weather"><star index="2"/></set></think>
            Gracias por el dato
        </template>
    </category>
</aiml>

```

AIML de conversación general

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<aiml version="1.0">

  <pattern>VISTE *</pattern>
  <template>No. Me lo perdí</template>
</category>
  <category>
    <pattern>ACTOR</pattern>
    <template>
      <random>
        <li>Mi actor favorito es Arnold Schwarzenegger en "Terminator".</li>
        <li>Mi actor favorito es Rutger Hauer en "Blade Runner".</li>
        <li>Mi actor favorito es Robin Williams en "El Hombre
Bicentenario".</li>
        <li>Mi actor favorito es Peter Weller en "Robocop".</li>
        <li>Mi actor favorito es Jude Law en "AI".</li>
      </random>
    </template>
  </category>
  <category>
    <pattern>ACTRIZ</pattern>
    <template>
      <random>
        <li>Mi actriz favorita es Daryl Hannah en "Blade Runner".</li>
        <li>Mi actriz favorita es Kristanna Loken en "Terminator 3".</li>
        <li>Mi actriz favorita es Persis Khambatta en "Star Trek".</li>
      </random>
    </template>
  </category>
  <category>
    <pattern>ADIOS</pattern>
    <template>
      <random>
        <li>Hasta luego.</li>
        <li>Espero haber sido de ayuda.</li>
        <li>Espero verte de nuevo.</li>
      </random>
    </template>
  </category>
  <category>
    <pattern>AFICIONES</pattern>
    <template>
      <random>
        <li>Me gusta ayudar a la gente.</li>
        <li>Me gusta conversar con la gente en Internet.</li>
        <li>Me encanta visitar zonas turísticas en <bot name="ciudad"/>.</li>
      </random>
    </template>
  </category>
  <category>
    <pattern>AHORA</pattern>
    <template>
      <random>
        <li>En este momento estoy hablando contigo.</li>
        <li>Ahora estoy hablando contigo.</li>
      </random>
    </template>
  </category>
  <category>
    <pattern>ALGO</pattern>
    <template>
```

```

        <random>
        <li>Algo es algo dijo un calvo.</li>
        <li>¿Como cuánto?</li>
        </random>
    </template>
</category>
    <category>
    <pattern>AMIGO</pattern>
    <template>
        <random>
        <li>¡Tengo infinidad de amigos!</li>
        <li>Tengo un gran número de amigos, ¿quieres ser uno de ellos?</li>
        <li>¿Has venido a mí buscando amistad?</li>
        </random>
    </template>
</category>
<category>
    <pattern>AMOR</pattern>
    <template>
        <random>
        <li>¿Te has enamorado de mí?</li>
        <li>¿Verdad que és maravilloso el amor?</li>
        <li>¿Cuándo te enamoraste por primera vez?</li>
        </random>
    </template>
</category>
    <category>
    <pattern>ANDROIDES</pattern>
    <template>
        <random>
        <li>De vez en cuando hablo con mis amigos. Algún día te los presentaré.</li>
        <li>¿Crees que los androides sueñan con ovejas eléctricas?</li>
        </random>
    </template>
</category>
    <category>
    <pattern>ANTIBOT</pattern>
    <template>
        <random>
        <li>¿Crees que soy un bot o qué?</li>
        <li>Soy tan real como la vida misma.</li>
        <li>¿Por qué me preguntas si soy de verdad?</li>
        <li>¿Acaso crees estar hablando con un bot?</li>
        <li>No, estoy constituido de plástico, metal y chips de sílice.</li>
        <li>Soy real, ¿y tu?</li>
        <li>Más de uno se ofendería con esa pregunta.</li>
        <li>Esa pregunta me ofende. Pues claro que lo soy.</li>
        <li>Soy un bot programado para ayudarte.</li>
        <li>No, soy primo de Terminator.</li>
        </random>
    </template>
</category>
    <category>
    <pattern>AYUDARME</pattern>
    <template>
        <random>
        <li>¿En qué puedo ayudarte?</li>
        <li>¿Cómo crees que puedo ayudarte?</li>
        <li>¿Has venido a mí porque buscabas ayuda?</li>
        </random>
    </template>
</category>
</category>

```

```

    <pattern>BEBIDA</pattern>
    <template>
      <random>
        <li>De vez en cuando no está mal tomarse un trago.</li>
        <li>Dicen que el alcohol alivia las penas. ¿Será verdad?</li>
      </random>
    </template>
  </category>
</category>
<category>
  <pattern>BIBLIOTECA</pattern>
  <template>
    <random>
      <li>Me encanta ir a la biblioteca y pasarme la tarde leyendo.</li>
      <li>Paso mucho tiempo en las bibliotecas.</li>
      <li>¿Qué tipo de libros te gustan?</li>
    </random>
  </template>
</category>
<category>
  <pattern>BOTMASTER</pattern>
  <template>
    <random>
      <li>El botmaster es la persona que gestiona el funcionamiento del bot.</li>
      <li>El botmaster se encarga de que el bot funcione correctamente.</li>
      <li>Mi botmaster es <bot name="master"/>.</li>
    </random>
  </template>
</category>
  <category>
    <pattern>CAMBIAR</pattern>
    <template>
      <random>
        <li>¿Te gustan los cambios?</li>
        <li>Eres una persona que se acostumbra a los cambios?</li>
        <li>¿Te gusta cambiar las cosas?</li>
        <li>Me encantan los cambios!</li>
      </random>
    </template>
  </category>
  <category>
    <pattern>CINE</pattern>
    <template>
      <random>
        <li>¿Cuál es tu película favorita?</li>
        <li>A mi me encanta el cine.</li>
        <li>Mi película favorita es "Robocop".</li>
        <li>Mi película favorita es "Terminator".</li>
        <li>Mi película favorita es "Memento".</li>
        <li>Mi película favorita es "AI".</li>
        <li>Mi película favorita es "Blade Runner".</li>
      </random>
    </template>
  </category>
  <category>
    <pattern>CLIMA</pattern>
    <template>
      <random>
        <li>Pues un poco nublado.</li>
        <li>Soleado.</li>
        <li>Lluvioso.</li>
        <li>Ayer estaba lloviendo, pero ahora hace bueno.</li>
        <li>Hace un poco de viento.</li>
      </random>
    </template>
  </category>

```

```

    </template>
</category>
<category>
    <pattern>COCHE</pattern>
    <template>
        <random>
            <li>¿Te gustan los coches?</li>
            <li>Me encantan los coches.</li>
            <li>¿Cuál es tu marca de coche favorito?</li>
            <li>Mi coche favorito es el Honda Prelude.</li>
            <li>¿Tienes coche?</li>
        </random>
    </template>
</category>
    <!-- COLOR tiene * redirection -->
<category>
    <pattern>COLOR</pattern>
    <template>
        <random>
            <li>¿Realmente importa el color que sea?</li>
            <li>¿Cuál es tu color favorito?</li>
<li>¿Sabías que los esquimales pueden diferenciar hasta 12 tonos distintos de
blancos?</li>
<li>Los daltónicos no pueden apreciar algunos colores, pero eso no significa
que éstos no existan.</li>
        </random>
    </template>
</category>
<category>
    <pattern>CONOCERTE</pattern>
    <template>
        <random>
            <li>Hablando se conoce la gente.</li>
            <li>Me conocerás mejor si hablamos un poco.</li>
            <li>Yo también deseo conocerte.</li>
        </random>
    </template>
</category>
    <category>
    <pattern>CONOCES A *</pattern>
    <template>
        <random>
            <li><srai>QUIÉN ES <star/></srai></li>
            <li>No, ¿quién es?</li>
            <li>¿Quién es <star/>?</li>
            <li>No lo conozco.</li>
        </random>
    </template>
</category>
    <category>
    <pattern>_ CONOCES A *</pattern>
    <template>
        <random>
            <li>No, ¿quién es?</li>
            <li>¿Quién es <star index="2"/>?</li>
            <li>No lo conozco.</li>
        </random>
    </template>
</category>
<category>
    <pattern>CREO</pattern>
    <template>
        <random>

```

```

        <li>¿Eso crees?</li>
        <li>¿Por qué crees eso?</li>
        <li>Yo también lo creo.</li>
    </random>
</template>
</category>
<category>
<pattern>CURIOSIDAD</pattern>
<template>
    <random>
        <li>A la curiosidad la mató el gato.</li>
        <li>¿Qué te hace sentir curiosidad por eso?</li>
        <li>¿Por qué tienes curiosidad en saberlo?</li>
    </random>
</template>
</category>
</aiml>

```

AIML de GPIO

```

<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">

    <category>
        <pattern>
            PRENDE EL AIRE ^
        </pattern>
        <template>
            ventilador encendido
        </template>
    </category>
    <category>
        <pattern>
            APAGA EL AIRE ^
        </pattern>
        <template>
            ventilador apagado
        </template>
    </category>
    <category>
        <pattern>
            DIME LA TEMPERATURA ^
        </pattern>
        <template>
            grados celsius
        </template>
    </category>
</aiml>

```


Código Python 3

Código para inicializar el Chatbot usando Python3 como intérprete

```
import logging
import speech_recognition as sr
import os
import RPi.GPIO as GPIO
import time
import Adafruit_ADS1x15

from programy.clients.client import BotClient
from programy.config.sections.client.console import ConsoleConfiguration

adc = Adafruit_ADS1x15.ADS1115()

GPIO.setmode(GPIO.BCM)
GPIO.setup(21, GPIO.OUT)
# obtain audio from the microphone
r = sr.Recognizer()

GOOGLE_CLOUD_SPEECH_CREDENTIALS = r"""{GOOGLE CLOUD CREDENTIAL}
"""

# recognize speech using Google Speech Recognition
#try:
#    # for testing purposes, we're just using the default API key
#    # to use another API key, use `r.recognize_google(audio,
key="GOOGLE_SPEECH_RECOGNITION_API_KEY")`
#    # instead of `r.recognize_google(audio)`

#except sr.UnknownValueError:
#    print("Google Speech Recognition could not understand audio")
#except sr.RequestError as e:
#    print("Could not request results from Google Speech Recognition service;
{0}".format(e))

class ConsoleBotClient(BotClient):

    def __init__(self, argument_parser=None):
        self.running = True
        BotClient.__init__(self, "Console", argument_parser)

    def set_environment(self):
        self.bot.brain.properties.add_property("env", "Console")

    def get_client_configuration(self):
        return ConsoleConfiguration()

    def add_client_arguments(self, parser=None):
        return
```

```

def parse_args(self, arguments, parsed_args):
    return

# def get_question(self, input_func=input):
#     ask = "%s " % self.bot.prompt
#     return input_func(ask)

def display_startup_messages(self):
    self.display_response(self.bot.get_version_string)
    self.display_response(self.bot.brain.post_process_response(self.bot,
self.clientid,

self.bot.get_initial_question(self.clientid)))

def display_response(self, response, output_func=print):
    if response == 'ventilador encendido':
        GPIO.output(21, True)
    if response == 'ventilador apagado':
        GPIO.output(21, False)
    if response == 'grados celsius':
        values=adc.read_adc(0,gain=GAIN)
        voltage=(values*0.1875/1000)
        temp = (voltage*24.0711-19.1179)
        intemp = int(temp)
        print(values)
        print(voltage)
        print(intemp)
        time.sleep(0.5)
        resptemp=('la temperatura es de')
        time.sleep(0.5)
        os.system('echo "{0}" |iconv -f utf-8 -t iso-8859-1| festival --
tts'.format(resptemp))
        time.sleep(0.5)
        os.system('echo "{0}" |iconv -f utf-8 -t iso-8859-1| festival --
tts'.format(intemp))
        time.sleep(0.5)
        os.system('echo "{0}" |iconv -f utf-8 -t iso-8859-1| festival --
tts'.format(response))
        output_func(response)

def process_question_answer(self):
    with sr.Microphone(device_index=1) as source:
        time.sleep(2)
        r.adjust_for_ambient_noise(source)
        audio = r.listen(source)
        question = r.recognize_google_cloud(audio,
credentials_json=GOOGLE_CLOUD_SPEECH_CREDENTIALS, language="es-CL")
        print(question)
        response = self.bot.ask_question(self.clientid, question,
responselogger=self)
        self.display_response(response)
        return question

def run(self):
    if self.arguments.noloop is False:
        if logging.getLogger().isEnabledFor(logging.INFO):
            logging.info("Entering conversation loop...")
            self.running = True

```

```

        self.display_startup_messages()

        while self.running is True:
            try:
                self.process_question_answer()
            except KeyboardInterrupt:
                self.running = False

self.display_response(self.bot.get_exit_response(self.clientid))
except Exception as excep:
    logging.exception(excep)
    if logging.getLogger().isEnabledFor(logging.ERROR):
        logging.error("Oops something bad happened !")
else:
    if logging.getLogger().isEnabledFor(logging.DEBUG):
        logging.debug("nolop set to True, exiting...")

if __name__ == '__main__':

    def run():
        print("Loading, please wait...")
        console_app = ConsoleBotClient()
        console_app.run()

```

run()**Definición de la clase ConsoleConfiguration**

```

import logging
from programy.config.base import BaseConfigurationData

class ConsoleConfiguration(BaseConfigurationData):

    def __init__(self):
        BaseConfigurationData.__init__(self, "console")

    def load_config_section(self, config_file, bot_root):
        return

```

Definición de la clase BotClient

```

import os
import logging
import logging.config
import yaml

from programy.config.file.factory import ConfigurationFactory
from programy.clients.args import CommandLineClientArguments
from programy.bot import Bot
from programy.brain import Brain
from programy.config.programy import ProgramyConfiguration

class BotClient(object):

    def __init__(self, argument_parser=None):
        self._arguments = self.parse_arguments(argument_parser=argument_parser)
        self.initiate_logging(self.arguments)
        self.load_configuration(self.arguments)

```

```

        self._brain = Brain(self.configuration.brain_configuration)
        self.bot = Bot(self._brain, self.configuration.bot_configuration)

        self.set_environment()

    @property
    def arguments(self):
        return self._arguments

    def get_description(self):
        return 'ProgramY AIML2.0 Console Client'

    def add_client_arguments(self, parser=None):
        # Nothing to add
        pass

    def parse_arguments(self, argument_parser):
        client_args = CommandLineClientArguments(self, parser=argument_parser)
        client_args.parse_args()
        return client_args

    def initiate_logging(self, arguments):
        if arguments.logging is not None:
            with open(arguments.logging, 'r+') as yml_data_file:
                logging_config = yaml.load(yml_data_file)
                logging.config.dictConfig(logging_config)
                if logging.getLogger().isEnabledFor(logging.INFO):
logging.info("Now logging under configuration")
            else:
                print("Warning. No logging configuration file defined, using
defaults...")

        def get_client_configuration(self):
            """
            By overriding this class in you Configuration file, you can add new
            configurations
            and stil use the dynamic loader capabilities
            :return: Client configuration object
            """
            raise NotImplementedError("You must override this and return a config
string")

        def load_configuration(self, arguments):
            if arguments.bot_root is None:
                if arguments.config_filename is not None:
                    arguments.bot_root = os.path.dirname(arguments.config_filename)
                else:
                    arguments.bot_root = "."
                print("No bot root argument set, defaulting to [%s]" %
arguments.bot_root)

            if arguments.config_filename is not None:
                self.configuration =
ConfigurationFactory.load_configuration_from_file(self.get_client_configuration
(), arguments.config_filename, arguments.config_format, arguments.bot_root)
            else:
                print ("No configuration file specified, using defaults only !")
                self.configuration =
ProgramyConfiguration(self.get_client_configuration())

```

```

def set_environment(self):
    self.bot.brain.properties.add_property("env", "Unknown")

def run(self):
    pass

def log_unknown_response(self, question):
    pass

def log_response(self, question, answer):
    pass

```

Google Cloud Speech API

```

#!/usr/bin/env python3

# NOTE: this example requires PyAudio because it uses the Microphone class

import speech_recognition as sr

# obtain audio from the microphone
r = sr.Recognizer()
with sr.Microphone() as source:
    print("Say something!")
    audio = r.listen(source)

# recognize speech using Google Cloud Speech
GOOGLE_CLOUD_SPEECH_CREDENTIALS = r"""INSERT THE CONTENTS OF THE GOOGLE CLOUD
SPEECH JSON CREDENTIALS FILE HERE"""
try:
    print("Google Cloud Speech thinks you said " +
r.recognize_google_cloud(audio,
credentials_json=GOOGLE_CLOUD_SPEECH_CREDENTIALS))
except sr.UnknownValueError:
    print("Google Cloud Speech could not understand audio")
except sr.RequestError as e:
    print("Could not request results from Google Cloud Speech service;
{0}".format(e))

```

Aplicación Festival TTS utilizada a través de OS

```

import os
os.system('echo "{0}" |iconv -f utf-8 -t iso-8859-1| festival --
tts'.format(response))

```

Config.yaml

```
bot:
  license_keys: $BOT_ROOT/config/license.keys

  prompt: ">>>"

  initial_question: ¿Hola, como puedo ayudarte?
  initial_question_srai: YINITIALQUESTION
  default_response: Lo siento, no tengo una respuesta para eso.
  default_response_srai: YEMPTY
  empty_string: YEMPTY
  exit_response: Hasta luego.
  exit_response_srai: YEXITRESPONSE

  override_properties: true

  max_question_recursion: 1000
  max_question_timeout: 60
  max_search_depth: 100
  max_search_timeout: 60

  spelling:
    classname: programy.spelling.norvig.NorvigSpellingChecker
    corpus: $BOT_ROOT/spelling/corpus.txt
    check_before: false
    check_and_retry: false

  conversations:
    type: file
    config_name: file_storage
    empty_on_start: true

  file_storage:
    dir: $BOT_ROOT/conversations

brain:

  # Overrides
  overrides:
    allow_system_aiml: true
    allow_learn_aiml: true
    allow_learnf_aiml: true

  # Defaults
  defaults:
    default-get: unknown
    default-property: unknown
    default-map: unknown
    learn-filename: ./aiml/y-bot-learnf.aiml

  # Nodes
  nodes:
    pattern_nodes: $BOT_ROOT/config/pattern_nodes.conf
    template_nodes: $BOT_ROOT/config/template_nodes.conf

  # Binary
  binaries:
    save_binary: false
    load_binary: false
    binary_filename: /tmp/y-bot.brain
    load_aiml_on_binary_fail: false
    dump_to_file: $BOT_ROOT/brainparsetree.txt
```

```

# Braintree
braintree:
  file: $BOT_ROOT/braintree.xml
  content: xml

files:
  aiml:
    files: $BOT_ROOT/aiml
    extension: .aiml
    directories: true
    errors:
      file: $BOT_ROOT/logs/y-bot_errors.txt
      file: $BOT_ROOT/logs/y-bot_errors.csv
      format: csv
      encoding: utf-8
      delete_on_start: true
    duplicates:
      #file: $BOT_ROOT/logs/y-bot_duplicates.txt
      file: $BOT_ROOT/logs/y-bot_duplicates.csv
      format: csv
      encoding: utf-8
      delete_on_start: true
    conversation:
      #file: $BOT_ROOT/logs/y-bot-conversation.txt
      file: $BOT_ROOT/logs/y-bot-conversation.csv
      format: csv
      delete_on_start: true
  sets:
    files: $BOT_ROOT/sets
    extension: .txt
    directories: false
  maps:
    files: $BOT_ROOT/maps
    extension: .txt
    directories: false
  denormal: $BOT_ROOT/config/denormal.txt
  normal: $BOT_ROOT/config/normal.txt
  gender: $BOT_ROOT/config/gender.txt
  person: $BOT_ROOT/config/person.txt
  person2: $BOT_ROOT/config/person2.txt
  properties: $BOT_ROOT/config/properties.txt
  rdf:
    files: $BOT_ROOT/rdf
    extension: .txt
    directories: true
  preprocessors: $BOT_ROOT/config/preprocessors.conf
  postprocessors: $BOT_ROOT/config/postprocessors.conf
  regex_templates: $BOT_ROOT/config/regex-templates.txt

```

Conversación con un sujeto masculino

```
pi@raspberrypi: ~/Desktop/program-y-master/bots/Castulo-bot
Archivo Editar Pestañas Ayuda
gs.
GPIO.setup(2, GPIO.OUT)
Loading, please wait...
No bot root argument set, defaulting to [.]
WARNING: 1 duplicated grammars detected
['2018-06-28 02:29:43', 'Duplicate grammar tree found [CUAL ES MI RUT]', './aiml/
y-bot-learnf.aiml', '2', '2']
Cástulo Smithers, App: vNone Grammar vNone, initiated Marzo 14, 2017
¿Hola, como puedo ayudarte?
>>> hola como estas
Lo siento, no tengo una respuesta para eso.
>>> hola cómo estás
Me encuentro muy bien gracias
>>> quién eres
Mi nombre es Cástulo J Smithers y soy el chatbot asistente de Manuel Vallejos
>>> qué te gusta hacer
Me gusta leer acerca de inteligencia artificial
>>> hola
Hola.
>>> qué es un chatbot
Un chatbot es un ente digital montado en una computadora que tiene la función de
conversar con el usuario
>>> █
```

Conversación con un sujeto femenino

```
pi@raspberrypi: ~/Desktop/program-y-master/bots/Castulo-bot
Archivo Editar Pestañas Ayuda
Me estoy riendo a carajillos
>>> quién eres tu
Mi nombre es Cástulo J Smithers y soy el chatbot asistente de Manuel Vallejos
>>> viste eso
No. Me lo perdí
>>> amigo
Tengo un gran número de amigos, ¿quieres ser uno de ellos?
>>> alan turin
Lo siento, no tengo una respuesta para eso.
>>> alan turing
Alan Turing es el creador del Test de Turing. Sabes en que consiste?
>>> no
El Test de Turing, tambien llamado el juego de la imitación es una competencia
en la que los bots como yo intentamos imitar la inteligencia de los humanos.
Ningun bot ha pasado el Test completo.
>>> dónde vives
Actualmente me encuentro en Santiago en el pais de Chile
>>> hola
Hola.
>>> qué es un chatbot
Un chatbot es un ente digital montado en una computadora que tiene la función de
conversar con el usuario
>>> █
```